

[illegible]

[illegible]

```
0001 0 MODULE STR$ARITH (
0002 0 IDENT = '1-019'
0003 0 ) =
0004 1 BEGIN
0005 1
0006 1 *****
0007 1 *
0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0010 1 * ALL RIGHTS RESERVED.
0011 1 *
0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0017 1 * TRANSFERRED.
0018 1 *
0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0021 1 * CORPORATION.
0022 1 *
0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0025 1 *
0026 1 *
0027 1 *****
0028 1
0029 1
0030 1 ++
0031 1 FACILITY: STRING Arithmetic
0032 1
0033 1 ABSTRACT:
0034 1
0035 1 This module is a large-precision arithmetic package based on
0036 1 decimal strings.
0037 1
0038 1 ENVIRONMENT: VAX-11 User Mode
0039 1
0040 1 AUTHOR: John Sauter, CREATION DATE: 01-MAR-1979
0041 1
0042 1 MODIFIED BY:
0043 1
0044 1 1-001 - Original. JBS 05-MAR-1979
0045 1 1-002 - Fix reciprocal of numbers between 0 and 1. JBS 07-MAR-1979
0046 1 1-003 - Treat minus 0 as zero. JBS 22-MAR-1979
0047 1 1-004 - Improve comments based on the code review. JBS 26-MAR-1979
0048 1 1-005 - Free local strings in case of an error. JBS 07-MAY-1979
0049 1 1-006 - Make the entry points take scalars by reference, in honor
0050 1 of the recognition of STR as a facility. JBS 15-MAY-1979
0051 1 1-007 - Change OTSS$ and LIBSS$ to STR$. JBS 21-MAY-1979
0052 1 1-008 - Restore some code deleted by mistake in edit 007.
0053 1 JBS 22-MAY-1979
0054 1 1-009 - Change calls to STR$COPY. JBS 16-JUL-1979
0055 1 1-010 - Correct a typo in a comment. JBS 30-JUL-1979
0056 1 1-011 - When freeing strings after an error, watch out for
0057 1 descriptors not yet initialized. JBS 31-JUL-1979
```



```
58 0058 1 1-012 - Added a new entry point - STR$DIVIDE. Also added related
59 0059 1 routines UPDATE_COUNTS, CVT_STR_PACKED and CVT_PACKED_STR.
60 0060 1 Also changed the existing string arithmetic routines, so
61 0061 1 that they all call LIB$ANALYZE_SDESC, to verify that the
62 0062 1 input descriptors are valid. LB 25-AUG-1981
63 0063 1 1-013 - Added updated code for STR$DIVIDE as well as ancillary
64 0064 1 routines UPDATE_COUNTS, CVT_STR_PACKED, and CVT_PACKED_STR.
65 0065 1 LB 16-NOV-81
66 0066 1 1-014 - Moved code to do the conversions to and from packed decimal
67 0067 1 into module LIBPKARIT. Changed code in STR$DIVIDE to use
68 0068 1 left justification of the input strings instead of right
69 0069 1 justification. RNH 11-DEC-81.
70 0070 1 1-015 - Added code in all string arithmetic routines so that they correctly
71 0071 1 handle all string classes. This required the addition of a new
72 0072 1 internal entry point CHK_STR_TYPE. LB 15-DEC-81.
73 0073 1 1-016 - Added code in STR$DIVIDE to zeroize a section of 8 bytes to avoid
74 0074 1 random data being picked up by the associated packed arithmetic
75 0075 1 routines that it calls. LB 1-APR-82.
76 0076 1 1-017 - Added code to ensure that a result of zero would always be returned
77 0077 1 as a positive value. LB 11-APR-82.
78 0078 1 1-018 - fixed call to LIB$FREE_VM at end of STR$DIVIDE to pass START_BUF
79 0079 1 by reference rather than by value. Initialize variable STORAGE
80 0080 1 on entry into STR$DIVIDE. Use its address rather than its contents
81 0081 1 when assigning QSTRBUF in special zero-quotient case. Allow for
82 0082 1 rounding to possibly propagate another digit into the quotient in
83 0083 1 STR$DIVIDE. Adjust calculation of BYTES_VM in STR$DIVIDE slightly
84 0084 1 to fix an access violation problem.
85 0085 1 MDL 11-Mar-1983
86 0086 1 1-019 - Enlarged amount of VM that STR$DIVIDE gets to prevent an
87 0087 1 access violation in certain cases. STAN 18-Jun-1984.
88 0088 1 --
89 0089 1
90 0090 1 !<BLF/PAGE>
```

```

92 0091 1 !+
93 0092 1 ! SWITCHES:
94 0093 1 !-
95 0094 1
96 0095 1 SWITCHES ADDRESSING_MODE (EXTERNAL = GENERAL, NONEXTERNAL = WORD_RELATIVE);
97 0096 1
98 0097 1 !+
99 0098 1 ! LINKAGES:
100 0099 1 !-
101 0100 1
102 0101 1 LINKAGE
103 0102 1 JSB1 = JSB (REGISTER=6, REGISTER=7) : NOPRESERVE (2,3,4,5),
104 0103 1 JSB2 = JSB (REGISTER=6, REGISTER=7, REGISTER=8) : NOPRESERVE (2,3,4,5),
105 0104 1 JSB3 = JSB (REGISTER=6, REGISTER=7, REGISTER=8, REGISTER=9)
106 0105 1 : NOPRESERVE (2,3,4,5),
107 0106 1 JSB4 = JSB (REGISTER=6, REGISTER=7, REGISTER=8, REGISTER=9, REGISTER=10)
108 0107 1 : NOPRESERVE (2,3,4,5);
109 0108 1
110 0109 1
111 0110 1 !+
112 0111 1 ! TABLE OF CONTENTS:
113 0112 1 !-
114 0113 1
115 0114 1 FORWARD ROUTINE
116 0115 1 STR$ADD : NOVALUE,
117 0116 1 STR$MUL : NOVALUE,
118 0117 1 STR$RECIP : NOVALUE,
119 0118 1 STR$ROUND : NOVALUE,
120 0119 1 STR$DIVIDE : NOVALUE,
121 0120 1 CHK_STR_TYPE : NOVALUE,
122 0121 1 FREE_STRINGS;
123 0122 1
124 0123 1 !+
125 0124 1 ! INCLUDE FILES:
126 0125 1 !-
127 0126 1
128 0127 1 REQUIRE 'RTLIN:RTLPSECT';
129 0222 1 LIBRARY 'RTLSTARLE';
130 0223 1
131 0224 1 !+
132 0225 1 ! MACROS:
133 0226 1
134 0227 1 NONE
135 0228 1 !-
136 0229 1
137 0230 1 !+
138 0231 1 ! PSECTS:
139 0232 1 !-
140 0233 1
141 0234 1 DECLARE_PSECTS (STR);
142 0235 1
143 0236 1 !+
144 0237 1 ! OWN STORAGE:
145 0238 1
146 0239 1 NONE
147 0240 1 !-

```

```

! Add two strings
! Multiply two strings
! Take the reciprocal of a string
! Round a string
! Divide two strings
! Check the string type
! Free local strings

```

```

! Macros for defining psects
! System definitions

```

```

! Declare psects for STR$ facility

```



```
149 0241 1 !+
150 0242 1 ! EXTERNAL REFERENCES:
151 0243 1 !-
152 0244 1
153 0245 1 EXTERNAL ROUTINE
154 0246 1 LIB$STOP, ! Signal fatal error
155 0247 1 STR$GET1_DX, ! Allocate a string
156 0248 1 STR$FREE1_DX, ! Deallocate a string
157 0249 1 STR$COPY_R, ! Copy a string by reference
158 0250 1 STR$COPY_DX, ! Copy a string by descriptor
159 0251 1 LIB$GET_VM, ! Allocate virtual memory
160 0252 1 LIB$FREE_VM, ! Deallocate virtual memory
161 0253 1 LIB$COPY_R_DX, ! Copy a string by reference
162 0254 1 LIB$ROUND_R7:JSB1 NOVALUE, ! Rounds quotient to correct length
163 0255 1 LIB$CALC_D_R7:JSB1, ! Calculates normalization factor
164 0256 1 LIB$CALC_Q_R9:JSB3, ! Calculates one quotient digit
165 0257 1 LIB$SUB_PACK_R8:JSB2, ! Subtracts two decimal arrays
166 0258 1 LIB$MUL_PACK_R10:JSB4 NOVALUE, ! Multiplies a packed array by a single
167 0259 1 ! entry
168 0260 1 LIB$ADJUST_Q_R9:JSB3 NOVALUE, ! Adjusts intermediate results of divi-
169 0261 1 ! sion algorithm if initial guess at
170 0262 1 ! a quotient digit is wrong
171 0263 1 LIB$CVT_STR_PACK_R9:JSB3 NOVALUE,
172 0264 1 ! Converts a string of decimal digits
173 0265 1 ! to an array of packed decimal
174 0266 1 ! values
175 0267 1 LIB$CVT_PACK_STR_R8:JSB2 NOVALUE,
176 0268 1 ! Converts an array of packed decimal
177 0269 1 ! values to a string
178 0270 1 LIB$ANALYZE_SDESC, ! Extract length and addr of a given
179 0271 1 ! descriptor and validate inputs
180 0272 1 LIB$MATCH_COND, ! Match condition codes
181 0273 1 STR$DUPL_CHAR; ! Used to pad result with leading zeroes
182 0274 1
183 0275 1 BIND
184 0276 1 ZERO = UPLIT BYTE (REP 7 OF (X'00'), X'0C'), ! Packed zero
185 0277 1 TEN = UPLIT BYTE (REP 6 OF (X'00'), X'01', X'0C'), ! Packed ten
186 0278 1 SPANC_TABLE = UPLIT BYTE (REP 48 OF (X'00'), REP 10 OF (X'01'),
187 0279 1 ! REP 198 OF (X'00')),
188 0280 1 MASK = UPLIT BYTE (REP 1 OF (X'01'));
189 0281 1
190 0282 1 BUILTIN
191 0283 1 CMPP, ! Compare packed decimal data
192 0284 1 MOVP, ! Move packed decimal data
193 0285 1 SPANC; ! Skip over a set of characters in a character string
194 0286 1
195 0287 1 !+
196 0288 1 ! The following are the error codes produced by this module.
197 0289 1 !-
198 0290 1
199 0291 1 EXTERNAL LITERAL
200 0292 1 LIB$INVARG, ! Invalid argument
201 0293 1 STR$DIVBY_ZER, ! Divide by zero.
202 0294 1 STR$WRONUMARG; ! Wrong number of arguments
203 0295 1
204 0296 1
```

```
206 0297 1 GLOBAL ROUTINE STR$ADD (
207 0298 1     ASIGN,
208 0299 1     AEXP,
209 0300 1     ADIGITS,
210 0301 1     BSIGN,
211 0302 1     BEXP,
212 0303 1     BDIGITS,
213 0304 1     CSIGN,
214 0305 1     CEXP,
215 0306 1     CDIGITS
216 0307 1 ) : NOVALUE =
217 0308 1
218 0309 1 ++
219 0310 1 FUNCTIONAL DESCRIPTION:
220 0311 1
221 0312 1     Add two decimal numbers. C := A + B
222 0313 1
223 0314 1 FORMAL PARAMETERS:
224 0315 1
225 0316 1     ASIGN.rv.r      0 = operand A is positive, 1 = negative
226 0317 1     AEXP.rl.r      Power of 10 by which to multiply the operand A
227 0318 1                  digits to get the absolute value of operand A.
228 0319 1                  E.g., AEXP = 1, ADIGITS = 123 gives 1230.
229 0320 1     ADIGITS.rnu.d    Descriptor for the digits of operand A
230 0321 1     BSIGN.rv.r      0 = operand B is positive, 1 = negative
231 0322 1     BEXP.rl.r      Power of 10 by which to multiply the operand B
232 0323 1                  digits to get the absolute value of operand B.
233 0324 1                  E.g., BEXP = -1, BDIGITS = 123 gives 12.3.
234 0325 1     BDIGITS.rnu.d    Descriptor for the digits of operand B
235 0326 1     CSIGN.wl.r      0 = operand C is positive, 1 = negative
236 0327 1     CEXP.wl.r      Power of 10 by which to multiply the operand C
237 0328 1                  digits to get the absolute value of operand C.
238 0329 1                  E.g., CEXP = 0, CDIGITS = 123 gives 123.
239 0330 1     CDIGITS.wnu.d    Descriptor for the digits of operand C
240 0331 1
241 0332 1 IMPLICIT INPUTS:
242 0333 1
243 0334 1     NONE
244 0335 1
245 0336 1 IMPLICIT OUTPUTS:
246 0337 1
247 0338 1     NONE
248 0339 1
249 0340 1 ROUTINE VALUE:
250 0341 1 COMPLETION CODES:
251 0342 1
252 0343 1     NONE
253 0344 1
254 0345 1 SIDE EFFECTS:
255 0346 1
256 0347 1     May allocate space for the CDIGITS string.
257 0348 1     Signals if storage is exceeded.
258 0349 1 --
259 0350 1
260 0351 2 BEGIN
261 0352 2
262 0353 2 MAP
```



```
263 0354 ADIGITS : REF BLOCK [8, BYTE],
264 0355 BDIGITS : REF BLOCK [8, BYTE],
265 0356 CDIGITS : REF BLOCK [8, BYTE],
266 0357
267 0358 LOCAL
268 0359
269 0360 + Internal form of A
270 0361 -
271 0362 A_DESC : BLOCK [8, BYTE] VOLATILE,
272 0363 ABUF : REF VECTOR [65535, BYTE],
273 0364 A_LEN,
274 0365 A_SIGN,
275 0366 + Internal form of B
276 0367 -
277 0368 B_DESC : BLOCK [8, BYTE] VOLATILE,
278 0369 BBUF : REF VECTOR [65535, BYTE],
279 0370 B_LEN,
280 0371 B_SIGN,
281 0372
282 0373 + Local copy of result.
283 0374 -
284 0375 RSIGN,
285 0376 REXP,
286 0377 R_DESC : BLOCK [8, BYTE] VOLATILE,
287 0378 RBUF : REF VECTOR [65535, BYTE],
288 0379 R_LEN,
289 0380 RESULT_DIGITS,
290 0381
291 0382 ! Addresses result
292 0383 ! Length of result
293 0384 ! Number of digits in result
294 0385
295 0386 + The following locals are needed for calls to LIB$ANALYZE_SDESC.
296 0387 -
297 0388 CBUF,
298 0389 C_LEN,
299 0390 STATUS;
300 0391
301 0392 BUILTIN
302 0393 ACTUALCOUNT;
303 0394
304 0395 + Enable a handler to free the local strings in case of an error.
305 0396 -
306 0397
307 0398 ENABLE
308 0399 FREE_STRINGS (A_DESC, B_DESC, R_DESC);
309 0400
310 0401 + Check for the proper number of arguments.
311 0402 -
312 0403
313 0404 IF (ACTUALCOUNT () LSS 9)
314 0405 THEN
315 0406 BEGIN
316 0407 LOCAL
317 0408 ROUT_NAME_DESC : BLOCK [9, BYTE];
318 0409
319 0410
```



```
0411 ROUT_NAME_DESC [DSC$W_LENGTH] = 7;
0412 ROUT_NAME_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_T;
0413 ROUT_NAME_DESC [DSC$B_CLASS] = DSC$K_CLASS_S;
0414 ROUT_NAME_DESC [DSC$A_POINTER] = UPLIT (XASCII'STR$ADD');
0415 LIB$STOP (STR$_WRONUMARG, 2, ACTUALCOUNT (), ROUT_NAME_DESC);
0416 END;
0417
0418
0419 +
0420 Copy the A and B operands, taking the tens complement of the negative
0421 ones.
0422 -
0423 A_DESC [DSC$W_LENGTH] = 0;
0424 A_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
0425 A_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
0426 A_DESC [DSC$A_POINTER] = 0;
0427 +
0428 Compute the length of operand A. Only the leading digits count.
0429 (Somday use SCAN or SPAN for this.)
0430 First call LIB$ANALYZE_SDESC to ensure that the input descriptor
0431 is valid. If it is, then ABUF will contain the address of the
0432 first byte of the string, and A_LEN will contain its length.
0433 -
0434
0435 STATUS = LIB$ANALYZE_SDESC (.ADIGITS,A_LEN,ABUF);
0436 IF .STATUS NEQ SSS_NORMAL
0437 THEN
0438     LIB$STOP (LIB$_INVARG);
0439
0440 +
0441 Check here for the CDIGITS descriptor before getting too involved
0442 in the routine.
0443 -
0444
0445 STATUS = LIB$ANALYZE_SDESC (.CDIGITS,C_LEN,CBUF);
0446 IF .STATUS NEQ SSS_NORMAL
0447 THEN
0448     LIB$STOP (LIB$_INVARG);
0449 A_LEN = 0;
0450 A_SIGN = ..ASIGN;
0451 BEGIN
0452
0453 LOCAL
0454     SCAN_DONE;
0455
0456 SCAN_DONE = 0;
0457
0458 DO
0459     BEGIN
0460         IF (.A_LEN EQLU .ADIGITS [DSC$W_LENGTH])
0461         THEN
0462             SCAN_DONE = 1
0463         ELSE
0464             IF ((.ABUF [.A_LEN] GEQ %C'0') AND (.ABUF [.A_LEN] LEQ %C'9'))
0465             THEN
```

```

377      0468 4      A_LEN = .A_LEN + 1
378      0469 4      ELSE
379      0470 4          SCAN_DONE = 1;
380      0471 4
381      0472 4      END
382      0473 4      UNTIL (.SCAN_DONE);
383      0474 4
384      0475 4      END;
385      0476 4      A_LEN = .A_LEN + 1;                ! Extra digit for sign
386      0477 4      STR$GET1 DX (A_LEN, A_DESC);
387      0478 4      ABUF = .A_DESC[DSC$A_POINTER];
388      0479 4      ABUF [0] = %C'0';
389      0480 4      CH$MOVE (.A_LEN - 1, .ADIGITS [DSC$A_POINTER], ABUF [1]);
390      0481 4
391      0482 4      IF (.A_SIGN)
392      0483 4      THEN
393      0484 4          BEGIN
394      0485 4      !+ Take the tens complement of the A operand. This is done by
395      0486 4      !- subtracting each digit from 9, and adding 1 to the result. The final
396      0487 4      !- add can cause carries.
397      0488 4
398      0489 4
399      0490 4          DECR COUNTER FROM .A_LEN - 1 TO 0 DO
400      0491 4              ABUF [.COUNTER] = (9 - (.ABUF [.COUNTER] - %C'0')) + %C'0';
401      0492 4
402      0493 4          BEGIN
403      0494 4
404      0495 4          LOCAL
405      0496 4              CARRY_DONE,
406      0497 4              CARRY_COUNTER;
407      0498 4
408      0499 4          CARRY_DONE = 0;
409      0500 4          CARRY_COUNTER = .A_LEN - 1;
410      0501 4
411      0502 4          IF (.CARRY_COUNTER GEQ 0)
412      0503 4          THEN
413      0504 4              DO
414      0505 4                  BEGIN
415      0506 4                      ABUF [.CARRY_COUNTER] = .ABUF [.CARRY_COUNTER] + 1;
416      0507 4
417      0508 4                      IF (.ABUF [.CARRY_COUNTER] LEQ %C'9')
418      0509 4                      THEN
419      0510 4                          CARRY_DONE = 1
420      0511 4                      ELSE
421      0512 4                          BEGIN
422      0513 4                              ABUF [.CARRY_COUNTER] = .ABUF [.CARRY_COUNTER] - 10;
423      0514 4                              CARRY_COUNTER = .CARRY_COUNTER - 1;
424      0515 4                              END;
425      0516 4
426      0517 4                      END;
427      0518 4
428      0519 4                  END
429      0520 4              UNTIL ((.CARRY_DONE) OR (.CARRY_COUNTER LSS 0));
430      0521 4
431      0522 4          IF ( NOT .CARRY_DONE) THEN A_SIGN = 0;
432      0523 4
433      0524 4      END;
```

```
434 0525      END;
435 0526
436 0527      B_DESC [DSC$W_LENGTH] = 0;
437 0528      B_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
438 0529      B_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
439 0530      B_DESC [DSC$A_POINTER] = 0;
440 0531
441 0532      * Compute the length of operand B. Only the leading digits count.
442 0533      First call LIB$ANALYZE_SDESC to ensure that the input descriptor
443 0534      is valid. If it is, then BBUF will contain the address of the
444 0535      first byte of the string, and B_LEN will contain its length.
445 0536
446 0537
447 0538      STATUS = LIB$ANALYZE_SDESC (.BDIGITS, B_LEN, BBUF);
448 0539      IF .STATUS NEQ SSS_NORMAL
449 0540      THEN
450 0541          LIB$STOP (LIB$INVARG);
451 0542      B_LEN = 0;
452 0543      B_SIGN = ..BSIGN;
453 0544      BEGIN
454 0545
455 0546      LOCAL
456 0547          SCAN_DONE;
457 0548
458 0549      SCAN_DONE = 0;
459 0550
460 0551      DO
461 0552          BEGIN
462 0553
463 0554              IF (.B_LEN EQLU .BDIGITS [DSC$W_LENGTH])
464 0555              THEN
465 0556                  SCAN_DONE = 1
466 0557              ELSE
467 0558
468 0559                  IF ((.BBUF [.B_LEN] GEQ %C'0') AND (.BBUF [.B_LEN] LEQ %C'9'))
469 0560                  THEN
470 0561                      B_LEN = .B_LEN + 1
471 0562                  ELSE
472 0563                      SCAN_DONE = 1;
473 0564
474 0565                  END
475 0566              UNTIL (.SCAN_DONE);
476 0567
477 0568      END;
478 0569      B_LEN = .B_LEN + 1;          ! Extra digit for sign
479 0570      STR$GET1_DX (B_LEN, B_DESC);
480 0571      BBUF = .B_DESC [DSC$A_POINTER];
481 0572      BBUF [0] = %C'0';
482 0573      CH$MOVE (.B_LEN - 1, .BDIGITS [DSC$A_POINTER], BBUF [1]);
483 0574
484 0575      IF (.B_SIGN)
485 0576      THEN
486 0577          BEGIN
487 0578
488 0579      * Take the tens complement of the B operand. This is done by
489 0580      subtracting each digit from 9, and adding 1 to the result. The final
490 0581      add can cause carries.
```



```
491 0582 !-
492 0583
493 0584
494 0585     DECR COUNTER FROM .B_LEN - 1 TO 0 DO
495 0586         BBUF [.COUNTER] = (9 - (.BBUF [.COUNTER] - %C'0')) + %C'0';
496 0587
497 0588     BEGIN
498 0589
499 0590     LOCAL
500 0591         CARRY_DONE
501 0592         CARRY_COUNTER;
502 0593
503 0594     CARRY_DONE = 0;
504 0595     CARRY_COUNTER = .B_LEN - 1;
505 0596
506 0597     IF (.CARRY_COUNTER GEQ 0)
507 0598     THEN
508 0599         DO
509 0600             BEGIN
510 0601                 BBUF [.CARRY_COUNTER] = .BBUF [.CARRY_COUNTER] + 1;
511 0602
512 0603                 IF (.BBUF [.CARRY_COUNTER] LEQ %C'9')
513 0604                 THEN
514 0605                     CARRY_DONE = 1
515 0606                 ELSE
516 0607                     BEGIN
517 0608                         BBUF [.CARRY_COUNTER] = .BBUF [.CARRY_COUNTER] - 10;
518 0609                         CARRY_COUNTER = .CARRY_COUNTER - 1;
519 0610                     END;
520 0611
521 0612                 END
522 0613             UNTIL ((.CARRY_DONE) OR (.CARRY_COUNTER LSS 0));
523 0614
524 0615     IF ( NOT .CARRY_DONE) THEN B_SIGN = 0;
525 0616
526 0617     END;
527 0618     END;
528 0619
529 0620     +
530 0621     Compute a tentative result exponent based on the smallest exponent
531 0622     in either A or B.
532 0623     -
533 0624     REXP = MIN (..AEXP, ..BEXP);
534 0625     +
535 0626     Allocate enough space to hold the maximum possible number of result
536 0627     digits. This is done by spanning the powers of ten involved in the
537 0628     two input operands, and adding 1 for carry.
538 0629     -
539 0630     RESULT_DIGITS = (MAX (..AEXP + .A_LEN, ..BEXP + .B_LEN)) + 1 - .REXP;
540 0631     R_DESC [DSC$W_LENGTH] = 0;
541 0632     R_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
542 0633     R_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
543 0634     R_DESC [DSC$A_POINTER] = 0;
544 0635     STR$GET1 DX (RESULT_DIGITS, R_DESC);
545 0636     RBUF = .R_DESC [DSC$A_POINTER];
546 0637     R_LEN = .R_DESC [DSC$W_LENGTH];
547 0638     !+
```

```
548 0639 2 | Copy the A operand into the result string, offsetting it properly
549 0640 2 | based on the exponents.
550 0641 2 |
551 0642 2 | CHSFILL (XC'0', .R_LEN, .R_DESC [DSC$A_POINTER]);
552 0643 2 | CHSMOVE (.A_LEN, .A_DESC [DSC$A_POINTER],
553 0644 2 | .R_DESC [DSC$A_POINTER] + .R_LEN - (.AEXP - .REXP) - .A_LEN);
554 0645 2 |
555 0646 2 | +
556 0647 2 | | If the A operand was negative we owe high-order nines.
557 0648 2 | |
558 0649 2 | | IF (.A_SIGN) THEN CHSFILL (XC'9', (.R_LEN - .A_LEN) - (.AEXP - .REXP), .R_DESC [DSC$A_POINTER]);
559 0650 2 | |
560 0651 2 | | +
561 0652 2 | | | Now add in the B operand.
562 0653 2 | | |
563 0654 2 | | | DECR COUNTER FROM (.R_LEN - 1 - (.BEXP - .REXP)) TO (.R_LEN - 1 - (.BEXP - .REXP) - (.B_LEN - 1)) DO
564 0655 2 | | | BEGIN
565 0656 2 | | |
566 0657 2 | | | LOCAL
567 0658 2 | | | B_INDEX,
568 0659 2 | | | SUM;
569 0660 2 | | |
570 0661 2 | | | B_INDEX = .COUNTER - (.R_LEN - 1 - (.BEXP - .REXP) - (.B_LEN - 1));
571 0662 2 | | | SUM = .RBUF [.COUNTER] + .BBUF [B_INDEX] - XC'0';
572 0663 2 | | |
573 0664 2 | | | IF (.SUM GTR XC'9')
574 0665 2 | | | THEN
575 0666 2 | | | BEGIN
576 0667 2 | | | +
577 0668 2 | | | | We must propagate a carry to the higher digits of RBUF
578 0669 2 | | | |
579 0670 2 | | | | LOCAL
580 0671 2 | | | | CARRY_DONE,
581 0672 2 | | | | CARRY_COUNTER;
582 0673 2 | | | |
583 0674 2 | | | | RBUF [.COUNTER] = .SUM - 10;
584 0675 2 | | | | CARRY_DONE = 0;
585 0676 2 | | | | CARRY_COUNTER = .COUNTER - 1;
586 0677 2 | | | |
587 0678 2 | | | | IF (.CARRY_COUNTER GEQ 0)
588 0679 2 | | | | THEN
589 0680 2 | | | | DO
590 0681 2 | | | | BEGIN
591 0682 2 | | | | RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] + 1;
592 0683 2 | | | |
593 0684 2 | | | | IF (.RBUF [.CARRY_COUNTER] LEQ XC'9')
594 0685 2 | | | | THEN
595 0686 2 | | | | CARRY_DONE = 1
596 0687 2 | | | | ELSE
597 0688 2 | | | | BEGIN
598 0689 2 | | | | RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] - 10;
599 0690 2 | | | | CARRY_COUNTER = .CARRY_COUNTER - 1;
600 0691 2 | | | | END;
601 0692 2 | | | |
602 0693 2 | | | |
603 0694 2 | | | |
604 0695 2 | | | |
```

```

605      0696      5      END
606      0697      4      UNTIL ((.CARRY_DONE) OR (.CARRY_COUNTER LSS 0));
607      0698      4
608      0699      4      END
609      0700      4      ELSE
610      0701      4      RBUF [.COUNTER] = .SUM;
611      0702      4
612      0703      4      END;
613      0704      4
614      0705      4      +
615      0706      4      End of the DECR loop.
616      0707      4      -
617      0708      4      +
618      0709      4      If the B operand is negative, we owe high-order nines.
619      0710      4      -
620      0711      3
621      0712      3      IF (.B_SIGN)
622      0713      3      THEN
623      0714      3      BEGIN
624      0715      3
625      0716      3      DECR COUNTER FROM ((.R_LEN - 1 - (.BEXP - .REXP) - (.B_LEN - 1)) - 1) TO 0 DO
626      0717      4      BEGIN
627      0718      4
628      0719      4      LOCAL
629      0720      4      SUM;
630      0721      4
631      0722      4      SUM = .RBUF [.COUNTER] + 9;
632      0723      4
633      0724      3      IF (.SUM GTR %C'9')
634      0725      4      THEN
635      0726      4      BEGIN
636      0727      4
637      0728      4      +
638      0729      4      We must propagate a carry to the higher digits of RBUF
639      0730      4      -
640      0731      4
641      0732      4      LOCAL
642      0733      4      CARRY_DONE,
643      0734      4      CARRY_COUNTER;
644      0735      4
645      0736      4      RBUF [.COUNTER] = .SUM - 10;
646      0737      4      CARRY_DONE = 0;
647      0738      4      CARRY_COUNTER = .COUNTER - 1;
648      0739      4
649      0740      4      IF (.CARRY_COUNTER GEQ 0)
650      0741      4      THEN
651      0742      4
652      0743      4      DO
653      0744      6      BEGIN
654      0745      6      RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] + 1;
655      0746      6
656      0747      6      IF (.RBUF [.CARRY_COUNTER] LEQ %C'9')
657      0748      6      THEN
658      0749      6      CARRY_DONE = 1
659      0750      6      ELSE
660      0751      7      BEGIN
661      0752      7      RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] - 10;
        CARRY_COUNTER = .CARRY_COUNTER - 1;

```



```
662      0753      6      END;  
663      0754  
664      0755      END  
665      0756      UNTIL ((.CARRY_DONE) OR (.CARRY_COUNTER LSS 0));  
666      0757  
667      0758      END  
668      0759      ELSE  
669      0760      RBUF [.COUNTER] = .SUM;  
670      0761  
671      0762      END;  
672      0763      END;  
673      0764  
674      0765  
675      0766  
676      0767      + Compute the sign of the result and recomplement it if negative.  
677      0768      -  
678      0769  
679      0770      IF (.RBUF [0] GEQ %C'5')  
680      0771      THEN  
681      0772      BEGIN  
682      0773      RSIGN = 1;  
683      0774  
684      0775      DECR COUNTER FROM .R_LEN - 1 TO 0 DO  
685      0776      RBUF [.COUNTER] = (9 - (.RBUF [.COUNTER] - %C'0')) + %C'0';  
686      0777  
687      0778      BEGIN  
688      0779      LOCAL  
689      0780      CARRY_DONE,  
690      0781      CARRY_COUNTER;  
691      0782  
692      0783      CARRY_DONE = 0;  
693      0784      CARRY_COUNTER = .R_LEN - 1;  
694      0785  
695      0786      IF (.CARRY_COUNTER GEQ 0)  
696      0787      THEN  
697      0788      DO  
698      0789      BEGIN  
699      0790      RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] + 1;  
700      0791  
701      0792      IF (.RBUF [.CARRY_COUNTER] LEQ %C'9')  
702      0793      THEN  
703      0794      CARRY_DONE = 1  
704      0795      ELSE  
705      0796      BEGIN  
706      0797      RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] - 10;  
707      0798      CARRY_COUNTER = .CARRY_COUNTER - 1;  
708      0799      END;  
709      0800  
710      0801      END;  
711      0802      END  
712      0803      UNTIL ((.CARRY_DONE) OR (.CARRY_COUNTER LSS 0));  
713      0804  
714      0805      END;  
715      0806      END  
716      0807      ELSE  
717      0808      RSIGN = 0;  
718      0809
```

```
719 0810 2
720 0811 2
721 0812 2
722 0813 2
723 0814 2
724 0815 2
725 0816 2
726 0817 2
727 0818 2
728 0819 2
729 0820 2
730 0821 2
731 0822 2
732 0823 2
733 0824 2
734 0825 2
735 0826 2
736 0827 2
737 0828 2
738 0829 2
739 0830 2
740 0831 2
741 0832 2
742 0833 2
743 0834 2
744 0835 2
745 0836 2
746 0837 2
747 0838 2
748 0839 2
749 0840 2
750 0841 2
751 0842 2
752 0843 2
753 0844 2
754 0845 2
755 0846 2
756 0847 2
757 0848 2
758 0849 2
759 0850 2
760 0851 2
761 0852 2
762 0853 2
763 0854 2
764 0855 2
765 0856 2
766 0857 2
767 0858 2
768 0859 2
769 0860 2
770 0861 2
771 0862 2
772 0863 2
773 0864 2
774 0865 2
775 0866 2

+ Discard low-order zeros, adjusting the exponent.
-
  BEGIN
    LOCAL
      SCAN_DONE
      SCAN_COUNTER;

    SCAN_DONE = 0;
    SCAN_COUNTER = .RESULT_DIGITS - 1;

    DO
      BEGIN
        IF (.SCAN_COUNTER LSS 0)
          THEN
            SCAN_DONE = 1
          ELSE
            IF (.RBUF [.SCAN_COUNTER] EQL %C'0') THEN SCAN_COUNTER = .SCAN_COUNTER - 1 ELSE SCAN_DONE = 1;

        END
      UNTIL (.SCAN_DONE);

    REXP = .REXP + ((.RESULT_DIGITS - 1) - .SCAN_COUNTER);
    RESULT_DIGITS = .SCAN_COUNTER + 1;
  END;

+ Remove high-order zeros.
-
  BEGIN
    LOCAL
      SCAN_DONE
      SCAN_COUNTER;

    SCAN_COUNTER = 0;
    SCAN_DONE = 0;

    DO
      BEGIN
        IF (.SCAN_COUNTER GEQ .RESULT_DIGITS)
          THEN
            SCAN_DONE = 1
          ELSE
            IF (.RBUF [.SCAN_COUNTER] EQL %C'0') THEN SCAN_COUNTER = .SCAN_COUNTER + 1 ELSE SCAN_DONE = 1;

        END
      UNTIL (.SCAN_DONE);

    IF (.SCAN_COUNTER GTR 0)
      THEN
```

```
776 0867      INCR COUNTER FROM 0 TO .RESULT_DIGITS - .SCAN_COUNTER - 1 DO
777 0868      RBUF [.COUNTER] = .RBUF [.COUNTER + .SCAN_COUNTER];
778 0869
779 0870      RESULT_DIGITS = .RESULT_DIGITS - .SCAN_COUNTER;
780 0871      END;
781 0872      +
782 0873      Return the results to the caller in the C operand.
783 0874      If there are no digits left, return a single zero digit.
784 0875      -
785 0876
786 0877      IF (.RESULT_DIGITS EQL 0)
787 0878      THEN
788 0879      BEGIN
789 0880      .CSIGN = 0;
790 0881      .CEXP = 0;
791 0882      STR$COPY_R (.CDIGITS, %REF (1), %REF (%ASCII'0'));
792 0883      CHK_STR_TYPE (.CDIGITS[DSC$A_POINTER],%REF (1),.CDIGITS);
793 0884      END
794 0885
795 0886      +
796 0887      Call CHK_STR_TYPE to determine if we need to pad the number with
797 0888      leading zeroes depending on the string type.
798 0889      -
799 0890
800 0891      ELSE
801 0892      BEGIN
802 0893      .CSIGN = .RSIGN;
803 0894      .CEXP = .REXP;
804 0895      CHK_STR_TYPE (.R_DESC[DSC$A_POINTER],RESULT_DIGITS,.CDIGITS);
805 0896      END;
806 0897
807 0898      ELSE
808 0899      BEGIN
809 0900      .CSIGN = .RSIGN;
810 0901      .CEXP = .REXP;
811 0902      STR$COPY_R (.CDIGITS, RESULT_DIGITS, .R_DESC [DSC$A_POINTER]);
812 0903      END;
813 0904
814 0905
815 0906      +
816 0907      Free our strings.
817 0908      -
818 0909      STR$FREE1_DX (R_DESC);
819 0910      STR$FREE1_DX (A_DESC);
820 0911      STR$FREE1_DX (B_DESC);
821 0912      END;
```

! end of STR\$ADD

.TITLE STR\$ARITH
.IDENT \1-019\

.PSECT _STR\$CODE,NOWRT, SHR, PIC,2

```
00# 00000 P.AAA: .BYTE 0[7]
0C 00007       .BYTE 12
00# 00008 P.AAB: .BYTE 0[6]
0C 01 0000E     .BYTE 1, 12
```

.....


```
00# 00010 P.AAC: .BYTE 0[48]
01# 00040 .BYTE 1[10]
00# 0004A P.AAD: .BYTE 0[198]
01 00110 .BYTE 1
    00111 .BLKB 3
00 44 44 41 24 52 54 53 00114 P.AAE: .ASCII \STR$ADD\<0>

ZERO= P.AAA
TEN= P.AAB
SPANC_TABLE= P.AAC
MASK= P.AAD

.EXTRN LIB$STOP, STR$GET1_DX
.EXTRN STR$FREE1_DX, STR$COPY_R
.EXTRN STR$COPY_DX, LIB$GET_VM
.EXTRN LIB$FREE_VM, LIB$COPY_R_DX
.EXTRN LIB$ROUND_R7, LIB$SCALE_D_R7
.EXTRN LIB$SCALE_Q_R9, LIB$SUB_PACK_R8
.EXTRN LIB$MUL_PACK_R10
.EXTRN LIB$ADJUST_Q-R9
.EXTRN LIB$CVT_STR_PACK_R9
.EXTRN LIB$CVT_PACK_STR_R8
.EXTRN LIB$ANALYZE_SDESC
.EXTRN LIB$MATCH_COND, STR$DUPL_CHAR
.EXTRN LIB$INVARG, STR$DIVBY_ZER
.EXTRN STR$WRONUMARG

OFFC 00000 .ENTRY STR$ADD, Save R2,R3,R4,R5,R6,R7,R8,R9,R10,- 0297
R11
MOVAB -68(SP), SP
CLRQ R_DESC 0351
CLRQ B_DESC
CLRQ A_DESC
MOVAL 59$, (FP)
CMPB (AP), #9 0405
BGEQU 1$
MOVL #17694727, ROUT_NAME_DESC 0412
MOVAB P.AAE, ROUT_NAME_DESC+4 0415
PUSHAB ROUT_NAME_DESC 0416
MOVZBL (AP), -(SP)
PUSHL #2
PUSHL #STR$WRONUMARG
CALLS #4, LIB$STOP
CLRQ A_DESC 0423
MOVB #T5, A_DESC+2 0424
MOVB #2, A_DESC+3 0425
CLRQ A_DESC+4 0426
PUSHAB ABUF 0435
PUSHAB A_LEN
MOVL ADIGITS, R2
PUSHL R2
CALLS #3, LIB$ANALYZE_SDESC
MOVL R0, STATUS
CML STATUS, #1 0436
BEQL 2$
PUSHL #LIB$INVARG 0438
CALLS #1, LIB$STOP
PUSHAB CBUF 0445
```

			14	AE	9F	00074	PUSHAB	C_LEN		
			24	AC	DD	00077	PUSHL	CDIGITS		
	00000000G	00		03	FB	0007A	CALLS	#3, LIB\$ANALYZE_SDESC		
		58		50	D0	00081	MOVL	R0, STATUS		
		01		58	D1	00084	CMPL	STATUS, #1		0446
				0D	13	00087	BEQL	3\$		
				8F	DD	00089	PUSHL	#LIB\$ INVARG		0448
	00000000G	00	00000000G	01	FB	0008F	CALLS	#1, LIB\$STOP		
				14	AE	D4	00096	CLRL	A_LEN	0449
	04	AE	04	BC	D0	00099	MOVL	@XSIGN, A_SIGN		0450
				51	D4	0009E	CLRL	SCAN_DONE		0456
14	AE	62	10	00	ED	000A0	CMPZV	#0, #16, (R2), A_LEN		0461
				15	13	000A6	BEQL	5\$		
	50	08	AE	14	AE	C1	000A8	ADDL3	A_LEN, ABUF, R0	0466
			30	60	91	000AE	CMPB	(R0), #48		
			39	0A	1F	000B1	BLSSU	5\$		
				60	91	000B3	CMPB	(R0), #57		
				05	1A	000B6	BGTRU	5\$		
				14	AE	D6	000B8	INCL	A_LEN	0468
				03	11	000BB	BRB	6\$		
		51		01	D0	000BD	MOVL	#1, SCAN_DONE		0470
		DD		51	E9	000C0	BLBC	SCAN_DONE, 4\$		0473
				14	AE	D6	000C3	INCL	A_LEN	0476
				3C	AE	9F	000C6	PUSHAB	A_DESC	0477
				18	AE	9F	000C9	PUSHAB	A_LEN	
	00000000G	00		02	FB	000CC	CALLS	#2, STR\$GET1 DX		
		08	AE	40	AE	D0	000D3	MOVL	A_DESC+4, ABUF	0478
			56	08	AE	D0	000D8	MOVL	ABUF, R6	0479
			66	30	90	000DC	MOVB	#48, (R6)		
	57	14	AE	01	C3	000DF	SUBL3	#1, A_LEN, R7		0480
01	A6	04	B2	57	28	000E4	MOVC3	R7, @Z(R2), 1(R6)		
			38	04	AE	E9	000EA	BLBC	A_SIGN, 13\$	0482
			50	01	A7	9E	000EE	MOVAB	1(R7), COUNTER	0491
				07	11	000F2	BRB	8\$		
6046		69	8F	6046	83	000F4	SUBB3	(COUNTER)[R6], #105, (COUNTER)[R6]		0492
			F6	50	F4	000FB	SOBGEQ	COUNTER, 7\$		
				51	D4	000FE	CLRL	CARRY_DONE		0500
			50	57	D0	00100	MOVL	R7, CARRY_COUNTER		0501
				18	19	00103	BLSS	12\$		0503
				6046	96	00105	INCB	(CARRY_COUNTER)[R6]		0508
		39		6046	91	00108	CMPB	(CARRY_COUNTER)[R6], #57		0510
				05	1A	0010C	BGTRU	10\$		
		51		01	D0	0010E	MOVL	#1, CARRY_DONE		0512
				06	11	00111	BRB	11\$		
		6046		0A	82	00113	SUBB2	#10, (CARRY_COUNTER)[R6]		0515
				50	D7	00117	DECL	CARRY_COUNTER		0516
		0A		51	E8	00119	BLBS	CARRY_DONE, 13\$		0520
				50	D5	0011C	TSTL	CARRY_COUNTER		
				E5	18	0011E	BGEQ	9\$		
		03		51	E8	00120	BLBS	CARRY_DONE, 13\$		0522
				04	AE	D4	00123	CLRL	A_SIGN	
			34	AE	B4	00126	CLRW	B_DESC		0527
	36	AE		0F	90	00129	MOVB	#T5, B_DESC+2		0528
	37	AE		02	90	0012D	MOVB	#2, B_DESC+3		0529
				38	AE	D4	00131	CLRL	B_DESC+4	0530
				18	AE	9F	00134	PUSHAB	BBUF	0538
				20	AE	9F	00137	PUSHAB	B_LEN	

		52	18	AC	D0	0013A	MOVL	BDIGITS, R2			
		00000000G	00	52	DD	0013E	PUSHL	R2			
			58	03	FB	00140	CALLS	#3, LIB\$ANALYZE_SDESC			
			01	50	D0	00147	MOVL	R0, STATUS			
				58	D1	0014A	CMPL	STATUS, #1		0539	
				0D	13	0014D	BEQL	14\$			
		00000000G	00	8F	DD	0014F	PUSHL	#LIB\$ INVARG		0541	
				01	FB	00155	CALLS	#1, LIB\$STOP			
			6E	1C	AE	D4	0015C	14\$: CLRL	B_LEN	0542	
				10	BC	D0	0015F	MOVL	@BSIGN, B_SIGN	0543	
					51	D4	00163	CLRL	SCAN_DONE	0549	
1C	AE	62	10	00	ED	00165	15\$: CMPZV	#0, #16, (R2), B_LEN		0554	
		50	18	AE	C1	0016D	BEQL	16\$			
				30	91	00173	ADDL3	B_LEN, BBUF, R0		0559	
					60	91	00176	CMPL	(R0), #48		
			39	0A	1F	00176	BLSSU	16\$			
					60	91	00178	CMPL	(R0), #57		
					05	1A	0017B	BGTRU	16\$		
				1C	AE	D6	0017D	INCL	B_LEN	0561	
					03	11	00180	BRB	17\$		
		51		01	D0	00182	16\$: MOVL	#1, SCAN_DONE		0563	
		DD		51	E9	00185	17\$: BLBC	SCAN_DONE, 15\$		0566	
				1C	AE	D6	00188	INCL	B_LEN	0569	
				34	AE	9F	0018B	PUSHAB	B_DESC	0570	
				20	AE	9F	0018E	PUSHAB	B_LEN		
		00000000G	00	02	FB	00191	CALLS	#2, STR\$GET1 DX			
			18	AE	D0	00198	MOVL	B_DESC+4, BBUF		0571	
				59	AE	D0	0019D	MOVL	BBUF, R9	0572	
				69	30	90	001A1	MOVB	#48, (R9)		
	5B	1C		01	C3	001A4	SUBL3	#1, B_LEN, R11		0573	
01	A9	04		5B	28	001A9	MOVC3	R11, #4(R2), 1(R9)			
				37	6E	E9	001AF	BLBC	B_SIGN, 24\$	0575	
				50	AB	9E	001B2	MOVAB	1(R11), COUNTER	0584	
					07	11	001B6	BRB	19\$		
	6049	69		8F	83	001B8	18\$: SUBB3	(COUNTER)[R9], #105, (COUNTER)[R9]		0585	
				F6	50	F4	001BF	19\$: SOBGEQ	COUNTER, 18\$		
					51	D4	001C2	CLRL	CARRY_DONE	0593	
				50	5B	D0	001C4	MOVL	R11, CARRY_COUNTER	0594	
					1B	19	001C7	BLSS	23\$	0596	
					6049	96	001C9	20\$: INCB	(CARRY_COUNTER)[R9]	0601	
				39	6049	91	001CC	CMPL	(CARRY_COUNTER)[R9], #57	0603	
					05	1A	001D0	BGTRU	21\$		
				51	01	D0	001D2	MOVL	#1, CARRY_DONE	0605	
					06	11	001D5	BRB	22\$		
		6049		0A	82	001D7	21\$: SUBB2	#10, (CARRY_COUNTER)[R9]		0608	
				50	D7	001DB	DECL	CARRY_COUNTER		0609	
				09	51	E8	001DD	22\$: BLBS	CARRY_DONE, 24\$	0613	
					50	D5	001E0	TSTL	CARRY_COUNTER		
					E5	18	001E2	BGEQ	20\$		
				02	51	E8	001E4	23\$: BLBS	CARRY_DONE, 24\$	0615	
					6E	D4	001E7	CLRL	B_SIGN		
				50	BC	D0	001E9	24\$: MOVL	@AEXP, R0	0624	
		14		BC	50	D1	001ED	CMPL	R0, @BEXP		
					04	15	001F1	BLEQ	25\$		
				50	BC	D0	001F3	MOVL	@BEXP, R0		
				57	50	D0	001F7	25\$: MOVL	R0, REXP		
	50	08		BC	14	AE	C1	001FA	ADDL3	A_LEN, @AEXP, R0	0630

51	14	BC	1C	AE	C1	00200	ADDL3	B_LEN, @BEXP, R1	
		51		50	D1	00206	CMPL	R0, R1	
				03	18	00209	BGEQ	26\$	
		50		51	D0	0020B	MOVL	R1, R0	
		50		57	C2	0020E	SUBL2	REXP, R0	
	20	AE	01	A0	9E	00211	MOVAB	1(R0), RESULT_DIGITS	
			2C	AE	B4	00216	CLRW	R_DESC	0631
	2E	AE		0F	90	00219	MOVB	#T5, R_DESC+2	0632
	2F	AE		02	90	0021D	MOVB	#2, R_DESC+3	0633
			30	AE	D4	00221	CLRL	R_DESC+4	0634
			2C	AE	9F	00224	PUSHAB	R_DESC	0635
			24	AE	9F	00227	PUSHAB	RESULT_DIGITS	
	00000000G	00		02	FB	0022A	CALLS	#2, STR\$GET1 DX	
		58	30	AE	D0	00231	MOVL	R_DESC+4, RBUF	0636
		56	2C	AE	3C	00235	MOVZWL	R_DESC, R_LEN	0637
56	30	6E		00	2C	00239	MOVCS	#0, (SP), #48, R_LEN, @R_DESC+4	0642
			30	BE		0023E			
	50	56	30	AE	C1	00240	ADDL3	R_DESC+4, R_LEN, R0	0644
	5A	57	08	BC	C3	00245	SUBL3	@AEXP, REXP, R10	
		50		5A	C0	0024A	ADDL2	R10, R0	
		50	14	AE	C2	0024D	SUBL2	A_LEN, R0	
	60	40	14	AE	28	00251	MOVCS	A_LEN, @A_DESC+4, (R0)	
			04	AE	E9	00257	BLBC	A_SIGN, 27\$	0649
	50	56	14	AE	C3	0025B	SUBL3	A_LEN, R_LEN, R0	
		50		5A	C0	00260	ADDL2	R10, R0	
50	39	6E		00	2C	00263	MOVCS	#0, (SP), #57, R0, @R_DESC+4	
			30	BE		00268			
	50	57	14	BC	C3	0026A	SUBL3	@BEXP, REXP, R0	0655
	51	56		50	C1	0026F	ADDL3	R0, R_LEN, R1	
	50	51		5B	C3	00273	SUBL3	R11, R1, R0	
		52	FF	A0	9E	00277	MOVAB	-1(R0), R2	
				45	11	0027B	BRB	33\$	0663
	50	51		52	C3	0027D	SUBL3	R2, COUNTER, B_INDEX	0662
		53		6148	9A	00281	MOVZBL	(COUNTER)[RBUF], R3	0663
		50		6049	9A	00285	MOVZBL	(B_INDEX)[R9], R0	
		50		53	C0	00289	ADDL2	R3, R0	
		50		30	C2	0028C	SUBL2	#48, SUM	
		39		50	D1	0028F	CMPL	SUM, #57	0665
				2A	15	00292	BLEQ	32\$	
6148		50		0A	83	00294	SUBB3	#10, SUM, (COUNTER)[RBUF]	0676
				53	D4	00299	CLRL	CARRY_DONE	0677
		50	FF	A1	9E	0029B	MOVAB	-1(R1), CARRY_COUNTER	0678
				21	19	0029F	BLSS	33\$	0680
				6048	96	002A1	INCB	(CARRY_COUNTER)[RBUF]	0685
		39		6048	91	002A4	CMPB	(CARRY_COUNTER)[RBUF], #57	0687
				05	1A	002A8	BGTRU	30\$	
		53		01	D0	002AA	MOVL	#1, CARRY_DONE	0689
				06	11	002AD	BRB	31\$	
	6048			0A	82	002AF	SUBB2	#10, (CARRY_COUNTER)[RBUF]	0692
				50	D7	002B3	DECL	CARRY_COUNTER	0693
	0A			53	E8	002B5	BLBS	CARRY_DONE, 33\$	0697
				50	D5	002B8	TSTL	CARRY_COUNTER	
				E5	18	002BA	BGEQ	29\$	
				04	11	002BC	BRB	33\$	0665
	6148			50	90	002BE	MOVB	SUM, (COUNTER)[RBUF]	0701
				51	D7	002C2	DECL	COUNTER	0655
	52			51	D1	002C4	CMPL	COUNTER, R2	

		B4	18	002C7	BGEQ	28\$		
42		6E	E9	002C9	BLBC	B SIGN, 40\$		0712
51		52	D0	002CC	MOVL	R2, COUNTER		0716
		3A	11	002CF	BRB	39\$		
50		6148	9A	002D1	MOVZBL	(COUNTER)[RBUF], SUM		0722
50		09	C0	002D5	ADDL2	#9, SUM		
39		50	D1	002D8	CMPL	SUM, #57		0724
		2A	15	002DB	BLEQ	38\$		
6148		50	0A	83	SUBB3	#10, SUM, (COUNTER)[RBUF]		0735
		52	D4	002E2	CLRL	CARRY_DONE		0736
50	FF	A1	9E	002E4	MOVAB	-1(R1), CARRY_COUNTER		0737
		21	19	002E8	BLSS	39\$		0739
		6048	96	002EA	INCB	(CARRY_COUNTER)[RBUF]		0744
39		6048	91	002ED	CMPB	(CARRY_COUNTER)[RBUF], #57		0746
		05	1A	002F1	BGTRU	36\$		
52		01	D0	002F3	MOVL	#1, CARRY_DONE		0748
		06	11	002F6	BRB	37\$		
6048		0A	82	002F8	SUBB2	#10, (CARRY_COUNTER)[RBUF]		0751
		50	D7	002FC	DECL	CARRY_COUNTER		0752
0A		52	E8	002FE	BLBS	CARRY_DONE, 39\$		0756
		50	D5	00301	TSTL	CARRY_COUNTER		
		E5	18	00303	BGEQ	35\$		
		04	11	00305	BRB	39\$		0724
6148		50	90	00307	MOVB	SUM, (COUNTER)[RBUF]		0760
C3		51	F4	0030B	SOBGEQ	COUNTER, 34\$		0716
35		68	91	0030E	CMPB	(RBUF), #53		0770
		37	1F	00311	BLSSU	46\$		
54		01	D0	00313	MOVL	#1, RSIGN		0773
50		56	D0	00316	MOVL	R_LEN, COUNTER		0775
		07	11	00319	BRB	42\$		
6048	69	8F	83	0031B	SUBB3	(COUNTER)[RBUF], #105, (COUNTER)[RBUF]		0776
		F6	50	F4	SOBGEQ	COUNTER, 41\$		
		51	D4	00325	CLRL	CARRY_DONE		0784
		50	A6	9E	MOVAB	-1(R6), CARRY_COUNTER		0785
			1F	19	BLSS	47\$		0787
		6048	96	0032D	INCB	(CARRY_COUNTER)[RBUF]		0792
39		6048	91	00330	CMPB	(CARRY_COUNTER)[RBUF], #57		0794
		05	1A	00334	BGTRU	44\$		
51		01	D0	00336	MOVL	#1, CARRY_DONE		0796
		06	11	00339	BRB	45\$		
6048		0A	82	0033B	SUBB2	#10, (CARRY_COUNTER)[RBUF]		0799
		50	D7	0033F	DECL	CARRY_COUNTER		0800
08		51	E8	00341	BLBS	CARRY_DONE, 47\$		0804
		50	D5	00344	TSTL	CARRY_COUNTER		
		E5	18	00346	BGEQ	43\$		
		02	11	00348	BRB	47\$		0770
		54	D4	0034A	CLRL	RSIGN		0809
		52	D4	0034C	CLRL	SCAN_DONE		0320
51	20	AE	01	C3	SUBL3	#1, RESULT DIGITS, R1		0821
		50	51	D0	MOVL	R1, SCAN_COUNTER		
			50	D5	TSTL	SCAN_COUNTER		0826
		0A	19	00358	BLSS	49\$		
30		6048	91	0035A	CMPB	(SCAN_COUNTER)[RBUF], #48		0831
		04	12	0035E	BNEQ	49\$		
		50	D7	00360	DECL	SCAN_COUNTER		
		03	11	00362	BRB	50\$		
52		01	D0	00364	MOVL	#1, SCAN_DONE		

	EC		52	E9	00367	50\$:	BLBC	SCAN_DONE, 48\$	0834
	51		50	C2	0036A		SUBL2	SCAN_COUNTER, R1	0836
	57		51	C0	0036D		ADDL2	R1, REXP	
20	AE	01	A0	9E	00370		MOVAB	1(R0), RESULT_DIGITS	0837
			50	7C	00375		CLRQ	SCAN_DONE	0849
20	AE		51	D1	00377	51\$:	CMPL	SCAN_COUNTER, RESULT_DIGITS	0854
			0A	18	0037B		BGEQ	52\$	
30			6148	91	0037D		CMPE	(SCAN_COUNTER)[RBUF], #48	0859
			04	12	00381		BNEQ	52\$	
			51	D6	00383		INCL	SCAN_COUNTER	
			03	11	00385		BRB	53\$	
	50		01	D0	00387	52\$:	MOVL	#1, SCAN_DONE	
	EA		50	E9	0038A	53\$:	BLBC	SCAN_DONE, 51\$	0862
			51	D5	0038D		TSTL	SCAN_COUNTER	0864
			17	15	0038F		BLEQ	56\$	
53	20	AE	51	C3	00391		SUBL3	SCAN_COUNTER, RESULT_DIGITS, R3	0867
		50	01	CE	00396		MNEGL	#1, COUNTER	
			09	11	00399		BRB	55\$	
52		50	51	C1	0039B	54\$:	ADDL3	SCAN_COUNTER, COUNTER, R2	0868
	6048		6248	9C	0039F		MOVB	(R2)[RBUF], (COUNTER)[RBUF]	
F3		50	53	F2	003A4	55\$:	AOBLSS	R3, COUNTER, 54\$	
	20	AE	51	C2	003A8	56\$:	SUBL2	SCAN_COUNTER, RESULT_DIGITS	0870
			31	12	003AC		BNEQ	57\$	0877
		1C	BC	D4	003AE		CLRL	@CSIGN	0880
		20	BC	D4	003B1		CLRL	@CEXP	0881
	04	AE	30	D0	003B4		MOVL	#48, 4(SP)	0882
	04	AE	04	AE	9F	003B8	PUSHAB	4(SP)	
			01	D0	003BB		MOVL	#1, 4(SP)	
		04	AE	9F	003BF		PUSHAB	4(SP)	
00000000G	00		24	AC	DD	003C2	PUSHL	CDIGITS	
			03	FB	003C5		CALLS	#3, STR\$COPY_R	
	08	AE	24	AC	DD	003CC	PUSHL	CDIGITS	0883
			01	D0	003CF		MOVL	#1, 8(SP)	
52	24	AC	08	AE	9F	003D3	PUSHAB	8(SP)	
			04	C1	003D6		ADDL3	#4, CDIGITS, R2	
			62	DD	003DB		PUSHL	(R2)	
			11	11	003DD		BRB	58\$	
	1C	BC	54	D0	003DF	57\$:	MOVL	RSIGN, @CSIGN	0893
	20	BC	57	D0	003E3		MOVL	REXP, @CEXP	0894
			24	AC	DD	003E7	PUSHL	CDIGITS	0895
			24	AE	9F	003EA	PUSHAB	RESULT_DIGITS	
			38	AE	DD	003ED	PUSHL	R_DESC#4	
0000V	CF		03	FB	003F0	58\$:	CALLS	#3, CHK_STR_TYPE	
			2C	AE	9F	003F5	PUSHAB	R_DESC	0909
00000000G	00		01	FB	003F8		CALLS	#T, STR\$FREE1_DX	
			3C	AE	9F	003FF	PUSHAB	A_DESC	0910
00000000G	00		01	FB	00402		CALLS	#T, STR\$FREE1_DX	
			34	AE	9F	00409	PUSHAB	B_DESC	0911
00000000G	00		01	FB	0040C		CALLS	#T, STR\$FREE1_DX	
			04	00413			RET		0912
			0000	00414	59\$:		.WORD	Save nothing	0351
	50		08	AC	D0	00416	MOVL	8(AP), R0	
	50		04	AD	D0	0041A	MOVL	4(R0), R0	
			E8	AD	9F	0041E	PUSHAB	R_DESC	
			F0	AD	9F	00421	PUSHAB	B_DESC	
			F8	AD	9F	00424	PUSHAB	A_DESC	
			03	DD	00427		PUSHL	#3	

D 12
 16-Sep-1984 01:27:51 YAX-11 BLISS-32 V4.0-742
 14-Sep-1984 12:40:01 [LIBRTL.SRC]STRARITH.B32;1

Page 22
(4)

0000V	7E CF	04	5E AC 03	DD 00429 7D 0042B FB 0042F 04 00434	PUSHL MOVQ CALLS RET	SP 4(AP) -(SP) #3, FREE_STRINGS
-------	----------	----	----------------	--	-------------------------------	---------------------------------------

: 822 0913 1

STR
1-0

.....

```
0914 1 GLOBAL ROUTINE STRSMUL (
0915 1     ASIGN,
0916 1     AEXP,
0917 1     ADIGITS,
0918 1     BSIGN,
0919 1     BEXP,
0920 1     BDIGITS,
0921 1     CSIGN,
0922 1     CEXP,
0923 1     CDIGITS
0924 1 ) : NOVALUE =
0925 1
0926 1 **
0927 1 FUNCTIONAL DESCRIPTION:
0928 1
0929 1     Multiply two decimal numbers. C := A * B
0930 1
0931 1 FORMAL PARAMETERS:
0932 1
0933 1     ASIGN.rv.r      0 = operand A is positive, 1 = negative
0934 1     AEXP.rl.r      Power of 10 by which to multiply the operand A
0935 1                    digits to get the absolute value of operand A.
0936 1                    E.g., AEXP = 1, ADIGITS = 123 gives 1230.
0937 1     ADIGITS.rnu.d   Descriptor for the digits of operand A
0938 1     BSIGN.rv.r      0 = operand B is positive, 1 = negative
0939 1     BEXP.rl.r      Power of 10 by which to multiply the operand B
0940 1                    digits to get the absolute value of operand B.
0941 1                    E.g., BEXP = -1, BDIGITS = 123 gives 12.3.
0942 1     BDIGITS.rnu.d   Descriptor for the digits of operand B
0943 1     CSIGN.wl.r      0 = operand C is positive, 1 = negative
0944 1     CEXP.wl.r      Power of 10 by which to multiply the operand C
0945 1                    digits to get the absolute value of operand C.
0946 1                    E.g., CEXP = 0, CDIGITS = 123 gives 123.
0947 1     CDIGITS.wnu.d   Descriptor for the digits of operand C
0948 1
0949 1 IMPLICIT INPUTS:
0950 1
0951 1     NONE
0952 1
0953 1 IMPLICIT OUTPUTS:
0954 1
0955 1     NONE
0956 1
0957 1 ROUTINE VALUE:
0958 1 COMPLETION CODES:
0959 1
0960 1     NONE
0961 1
0962 1 SIDE EFFECTS:
0963 1
0964 1     May allocate space for the CDIGITS string.
0965 1     Signals if storage is exhausted.
0966 1
0967 1 --
0968 1 BEGIN
0969 1
0970 1 MAP
```

```

881      0971      ADIGITS : REF BLOCK [8, BYTE],
882      0972      BDIGITS : REF BLOCK [8, BYTE],
883      0973      CDIGITS : REF BLOCK [8, BYTE];
884      0974
885      0975      LOCAL
886      0976      +
887      0977      Internal form of A.
888      0978      -
889      0979      A_DESC : BLOCK [8, BYTE] VOLATILE,
890      0980      A_BUF : REF VECTOR [65535, BYTE],
891      0981      A_LEN,
892      0982      A_SIGN,
893      0983      +
894      0984      Internal form of B.
895      0985      -
896      0986      B_DESC : BLOCK [8, BYTE] VOLATILE,
897      0987      B_BUF : REF VECTOR [65535, BYTE],
898      0988      B_LEN,
899      0989      B_SIGN,
900      0990      +
901      0991      Local copy of result.
902      0992      -
903      0993      R_SIGN,
904      0994      R_EXP,
905      0995      R_DESC : BLOCK [8, BYTE] VOLATILE,
906      0996      R_BUF : REF VECTOR [65535, BYTE],
907      0997      R_LEN,
908      0998
909      0999      +
910      1000      The following are locals for the call to LIB$ANALYZE_SDESC.
911      1001      -
912      1002      C_BUF,
913      1003      C_LEN,
914      1004      STATUS;
915      1005
916      1006      BUILTIN
917      1007      ACTUALCOUNT;
918      1008
919      1009      +
920      1010      Enable a handler to free the local strings in case of an error.
921      1011      -
922      1012
923      1013      ENABLE
924      1014      FREE_STRINGS (A_DESC, B_DESC, R_DESC);
925      1015
926      1016      +
927      1017      Check the number of arguments.
928      1018      -
929      1019
930      1020      IF (ACTUALCOUNT () LSS 9)
931      1021      THEN
932      1022      BEGIN
933      1023
934      1024      LOCAL
935      1025      ROUT_NAME_DESC : BLOCK [8, BYTE];
936      1026
937      1027      ROUT_NAME_DESC [DSC$W_LENGTH] = 7;

```



```
938 1028 3 ROUT_NAME_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_T;  
939 1029 3 ROUT_NAME_DESC [DSC$B_CLASS] = DSC$K_CLASS_S;  
940 1030 3 ROUT_NAME_DESC [DSC$A_POINTER] = UPLIT (XASC) I'STR$MUL';  
941 1031 3 LIB$STOP (STR$WRONUMARG, 2, ACTUALCOUNT (), ROUT_NAME_DESC);  
942 1032 3 END;  
943 1033 3  
944 1034 3 +  
945 1035 3 Copy the A and B operands.  
946 1036 3 -  
947 1037 3 A_DESC [DSC$W_LENGTH] = 0;  
948 1038 3 A_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;  
949 1039 3 A_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;  
950 1040 3 A_DESC [DSC$A_POINTER] = 0;  
951 1041 3 +  
952 1042 3 Compute the length of operand A. Only the leading digits count.  
953 1043 3 First call LIB$ANALYZE_SDESC to ensure that the input descriptor  
954 1044 3 is valid. If it is, then ABUF will contain the address of the  
955 1045 3 first byte of the string, and A_LEN will contain its length.  
956 1046 3 -  
957 1047 3  
958 1048 3 STATUS = LIB$ANALYZE_SDESC (.ADIGITS,A_LEN,ABUF);  
959 1049 3 IF .STATUS NEQ SS$NORMAL  
960 1050 3 THEN  
961 1051 3 LIB$STOP (LIB$INVARG);  
962 1052 3  
963 1053 3 +  
964 1054 3 Check here also for the CDIGITS descriptor before we get too  
965 1055 3 involved in the routine.  
966 1056 3 -  
967 1057 3  
968 1058 3 STATUS = LIB$ANALYZE_SDESC (.CDIGITS,C_LEN,CBUF);  
969 1059 3 IF .STATUS NEQ SS$NORMAL  
970 1060 3 THEN  
971 1061 3 LIB$STOP (LIB$INVARG);  
972 1062 3  
973 1063 3 A_LEN = 0;  
974 1064 3 A_SIGN = ..ASIGN;  
975 1065 3 BEGIN  
976 1066 3  
977 1067 3 LOCAL  
978 1068 3 SCAN_DONE;  
979 1069 3  
980 1070 3 SCAN_DONE = 0;  
981 1071 3  
982 1072 3 DO  
983 1073 3 BEGIN  
984 1074 3  
985 1075 3 IF (.A_LEN EQLU .ADIGITS [DSC$W_LENGTH])  
986 1076 3 THEN  
987 1077 3 SCAN_DONE = 1  
988 1078 3 ELSE  
989 1079 3  
990 1080 3 IF ((.ABUF [.A_LEN] GEQ X('0')) AND (.ABUF [.A_LEN] LEQ X('9'))  
991 1081 3 THEN  
992 1082 3 A_LEN = .A_LEN + 1  
993 1083 3 ELSE  
994 1084 3 SCAN_DONE = 1;
```

```

995 1085 4
996 1086 4
997 1087 4
998 1088 4
999 1089 4
1000 1090 4
1001 1091 4
1002 1092 4
1003 1093 4
1004 1094 4
1005 1095 4
1006 1096 4
1007 1097 4
1008 1098 4
1009 1099 4
1010 1100 4
1011 1101 4
1012 1102 4
1013 1103 4
1014 1104 4
1015 1105 4
1016 1106 4
1017 1107 4
1018 1108 4
1019 1109 4
1020 1110 4
1021 1111 4
1022 1112 4
1023 1113 4
1024 1114 4
1025 1115 4
1026 1116 4
1027 1117 4
1028 1118 4
1029 1119 4
1030 1120 5
1031 1121 4
1032 1122 4
1033 1123 4
1034 1124 4
1035 1125 5
1036 1126 4
1037 1127 4
1038 1128 4
1039 1129 4
1040 1130 4
1041 1131 4
1042 1132 4
1043 1133 4
1044 1134 4
1045 1135 4
1046 1136 4
1047 1137 4
1048 1138 4
1049 1139 4
1050 1140 4
1051 1141 4

      END
      UNTIL (.SCAN_DONE);

      END;
      STR$GET1 DX (A_LEN, A_DESC);
      ABUF = .A_DESC[DSC$A_POINTER];
      CH$MOVE (.A_LEN, .BDIGITS [DSC$A_POINTER], ABUF [0]);
      B_DESC [DSC$W_LENGTH] = 0;
      B_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
      B_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
      B_DESC [DSC$A_POINTER] = 0;

      +
      Compute the length of operand B. Only the leading digits count.
      First call LIB$ANALYZE SDESC to ensure that the input descriptor
      is valid. If it is, then BBUF will contain the address of the
      first byte of the string, and B_LEN will contain its length.
      -

      STATUS = LIB$ANALYZE SDESC (.BDIGITS, B_LEN, BBUF);
      IF .STATUS NEQ SS$NORMAL
      THEN
        LIB$STOP (LIB$INVARG);
      B_LEN = 0;
      B_SIGN = .BSIGN;
      BEGIN

      LOCAL
        SCAN_DONE;

      SCAN_DONE = 0;

      DO
        BEGIN
          IF (.B_LEN EQLU .BDIGITS [DSC$W_LENGTH])
          THEN
            SCAN_DONE = 1
          ELSE
            IF ((.BBUF [.B_LEN] GEQ %C'0') AND (.BBUF [.B_LEN] LEQ %C'9'))
            THEN
              B_LEN = .B_LEN + 1
            ELSE
              SCAN_DONE = 1;

          END
        UNTIL (.SCAN_DONE);

      END;
      STR$GET1 DX (B_LEN, B_DESC);
      BBUF = .B_DESC[DSC$A_POINTER];
      CH$MOVE (.B_LEN, .BDIGITS [DSC$A_POINTER], BBUF [0]);

      +
      Set the accumulator to zero.
      -
      R_DESC [DSC$W_LENGTH] = 0;
```

```
1052 1142 2 R_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;  
1053 1143 R_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;  
1054 1144 R_DESC [DSC$A_POINTER] = 0;  
1055 1145 STR$GET1_DX (XREF (1), R_DESC);  
1056 1146 RBUF = .R_DESC [DSC$A_POINTER];  
1057 1147 R_LEN = .R_DESC [DSC$W_LENGTH];  
1058 1148 RBUF [0] = %C'0';  
1059 1149 RSIGN = 0;  
1060 1150 REXP = 0;  
1061 1151  
1062 1152 + Go through each digit of B, adding appropriately shifted A to  
1063 1153 R the indicated number of times. This is like the old mechanical  
1064 1154 adding machines.  
1065 1155  
1066 1156  
1067 1157 INCR POS FROM 0 TO .B_LEN - 1 DO  
1068 1158 BEGIN  
1069 1159 LOCAL  
1070 1160 DIGIT;  
1071 1161  
1072 1162 DIGIT = .BBUF [(B_LEN - 1) - .POS];  
1073 1163  
1074 1164  
1075 1165 DECR COUNTER FROM .DIGIT TO %C'1' DO  
1076 1166 STR$ADD (XREF (0), POS, A_DESC, RSIGN, REXP, R_DESC, RSIGN, REXP, R_DESC);  
1077 1167  
1078 1168 END;  
1079 1169  
1080 1170  
1081 1171 + Compute the exponent and sign of the result.  
1082 1172  
1083 1173 REXP = .REXP + (.AEXP + .BEXP);  
1084 1174 RSIGN = (IF (.A_SIGN EQL .B_SIGN) THEN 0 ELSE 1);  
1085 1175  
1086 1176 + Return the result to the caller. Because it is the output of STR$ADD  
1087 1177 it is already in normal form.  
1088 1178  
1089 1179 .CSIGN = .RSIGN;  
1090 1180 .CEXP = .REXP;  
1091 1181  
1092 1182 +  
1093 1183 Call CHK_STR_TYPE to determine if we need to pad the number with  
1094 1184 leading zeroes depending on the string type.  
1095 1185  
1096 1186  
1097 1187 R_LEN = .R_DESC [DSC$W_LENGTH];  
1098 1188 CHK_STR_TYPE (.R_DESC [DSC$A_POINTER], R_LEN, .CDIGITS);  
1099 1189  
1100 1190  
1101 1191 STR$COPY_DX (.CDIGITS, R_DESC);  
1102 1192  
1103 1193 + Free our strings.  
1104 1194  
1105 1195 STR$FREE1_DX (R_DESC);  
1106 1196 STR$FREE1_DX (A_DESC);  
1107 1197 STR$FREE1_DX (B_DESC);  
1108 1198 END;
```

! end of STR\$MUL

00 4C 55 4D 24 52 54 53 00551 00554 P.AAF: .BLKB 3
.ASCII \STR\$MUL\<0>

					OFFC 00000		.ENTRY		
							STR\$MUL, Save R2,R3,R4,R5,R6,R7,R8,R9,R10,-		0914
							R11		
							STR\$GET1 DX, R11		
							#LIB\$ INVARG, R10		
							LIB\$ANALYZE \$DESC, R9		
							LIB\$STOP, R8		
							-76(SP), SP		
							R_DESC		0968
							B_DESC		
							A_DESC		
							17\$, (FP)		
							(AP), #9		1020
							1\$		
							#17694727, ROUT_NAME_DESC		1027
							P.AAF, ROUT_NAME_DESC+4		1030
							ROUT_NAME_DESC		1031
							(AP), -(SP)		
							#2		
							#STR\$ WRONUMARG		
							#4, LIB\$STOP		
							A_DESC		1037
							#15, A_DESC+2		1038
							#2, A_DESC+3		1039
							A_DESC+4		1040
							ABUF		1048
							A_LEN		
							ADIGITS, R2		
							R2		
							#3, LIB\$ANALYZE_SDESC		
							R0, STATUS		
							STATUS, #1		1049
							2\$		
							R10		1051
							#1, LIB\$STOP		
							CBUF		1058
							C_LEN		
							CDIGITS		
							#3, LIB\$ANALYZE_SDESC		
							R0, STATUS		
							STATUS, #1		1059
							3\$		
							R10		1061
							#1, LIB\$STOP		
							A_LEN		1063
							ASIGN, A_SIGN		1064
							SCAN_DONE		1070
							#0, #16, (R2), A_LEN		1075
							5\$		
							A_LEN, ABUF, R0		1080

			30	60	91	000AD	CMPB	(R0), #48	
				0A	1F	000B0	BLSSU	5\$	
			39	60	91	000B2	CMPB	(R0), #57	
				05	1A	000B5	BGTRU	5\$	
			10	AE	D6	000B7	INCL	A_LEN	1082
				03	11	000BA	BRB	6\$	
			51	01	D0	000BC	MOVL	#1, SCAN_DONE	1084
			DD	51	E9	000BF	BLBC	SCAN_DONE, 4\$	1087
				44	AE	9F	PUSHAB	A_DESC	1090
				14	AE	9F	PUSHAB	A_LEN	
			6B	02	FB	000C8	CALLS	#2, STR\$GET1 DX	
		04	AE	48	AE	D0	MOVL	A_DESC+4, ABOF	1091
04	BE	04	B2	10	AE	28	MOVC3	A_LEN, @4(R2), @ABUF	1092
				3C	AE	B4	CLRW	B_DESC	1093
		3E	AE	0F	90	000DA	MOVB	#T5, B_DESC+2	1094
		3F	AE	02	90	000DE	MOVB	#2, B_DESC+3	1095
				40	AE	D4	CLRL	B_DESC+4	1096
				14	AE	9F	PUSHAB	BBUF	1104
				1C	AE	9F	PUSHAB	B_LEN	
			52	18	AC	D0	MOVL	BDIGITS, R2	
					52	DD	PUSHL	R2	
			69	03	FB	000F1	CALLS	#3, LIB\$ANALYZE_SDESC	
			56	50	D0	000F4	MOVL	R0, STATUS	
			01	56	D1	000F7	CMPL	STATUS, #1	1105
				05	13	000FA	BEQL	7\$	
				5A	DD	000FC	PUSHL	R10	1107
			68	01	FB	000FE	CALLS	#1, LIB\$STOP	
				18	AE	D4	CLRL	B_LEN	1108
			56	10	BC	D0	MOVL	@BSIGN, B_SIGN	1109
					51	D4	CLRL	SCAN_DONE	1115
18	AE		10	00	ED	0010A	CMPL	#0, #16, (R2), B_LEN	1120
				15	13	00110	BEQL	9\$	
		50					ADDL3	B_LEN, BBUF, R0	1125
		14	AE	18	AE	C1	CMPB	(R0), #48	
			30	60	91	00118	BLSSU	9\$	
				0A	1F	0011B	CMPB	(R0), #57	
			39	60	91	0011D	BGTRU	9\$	
				05	1A	00120	INCL	B_LEN	1127
				18	AE	D6	BRB	10\$	
				03	11	00125	MOVL	#1, SCAN_DONE	1129
			51	01	D0	00127	BLBC	SCAN_DONE, 8\$	1132
			DD	51	E9	0012A	PUSHAB	B_DESC	1135
				3C	AE	9F	PUSHAB	B_LEN	
				1C	AE	9F	CALLS	#2, STR\$GET1 DX	
			6B	02	FB	00133	MOVL	B_DESC+4, BBUF	1136
		14	AE	40	AE	D0	MOVC3	B_LEN, @4(R2), @BBUF	1137
		04	B2	18	AE	28	CLRW	R_DESC	1141
				34	AE	B4	MOVB	#T5, R_DESC+2	1142
		36	AE	0F	90	00145	MOVB	#2, R_DESC+3	1143
		37	AE	02	90	00149	CLRL	R_DESC+4	1144
				38	AE	D4	PUSHAB	R_DESC	1145
				34	AE	9F	MOVL	#T, 4(SP)	
		04	AE	01	D0	00153	PUSHAB	4(SP)	
				04	AE	9F	CALLS	#2, STR\$GET1 DX	
			6B	02	FB	0015A	MOVL	R_DESC+4, RBUF	1146
			50	AE	D0	0015D	MOVZWL	R_DESC, R_LEN	1147
		28	AE	34	AE	3C	MOVB	#48, (RBUF)	1148
			60	30	90	00166			

			1C	AE	7C	00169	CLRG	REXP	1150
	24	AE	01	CE	0016C		MNEGL	#1, POS	1157
			3A	11	00170		BRB	14\$	
50	18	AE	24	AE	C3	00172	SUBL3	POS, B_LEN, RO	1163
		50	14	AE	C0	00178	ADDL2	BBUF, RO	
		52	FF	A0	9A	0017C	MOVZBL	-1(RO), DIGIT	
			25	11	00180		BRB	13\$	1165
			34	AE	9F	00182	PUSHAB	R_DESC	1166
			20	AE	9F	00185	PUSHAB	REXP	
			28	AE	9F	00188	PUSHAB	RSIGN	
			40	AE	9F	0018B	PUSHAB	R_DESC	
			2C	AE	9F	0018E	PUSHAB	REXP	
			34	AE	9F	00191	PUSHAB	RSIGN	
			5C	AE	9F	00194	PUSHAB	A_DESC	
			40	AE	9F	00197	PUSHAB	POS	
			20	AE	D4	0019A	CLRL	32(SP)	
			20	AE	9F	0019D	PUSHAB	32(SP)	
	FA1B	CF	09	FB	001A0		CALLS	#9, STR\$ADD	
			52	D7	001A5		DECL	COUNTER	
		31	52	D1	001A7	13\$:	CMPL	COUNTER, #49	
			D6	18	001AA		BGEQ	12\$	
CO	24	AE	18	AE	F2	001AC	AOBLSS	B_LEN, POS, 11\$	1157
50	08	BC	14	BC	C1	001B2	ADDL3	@BEXP, @AEXP, RO	1173
	1C	AE	50	C0	001B8		ADDL2	RO, REXP	
		56	57	D1	001BC		CMPL	A_SIGN, B_SIGN	1174
			04	12	001BF		BNEQ	15\$	
			50	D4	001C1		CLRL	RO	
			03	11	001C3		BRB	16\$	
		50	01	D0	001C5	15\$:	MOVL	#1, RO	
	20	AE	50	D0	001C8	16\$:	MOVL	RO, RSIGN	
	1C	BC	20	AE	D0	001CC	MOVL	RSIGN, @CSIGN	1179
	20	BC	1C	AE	D0	001D1	MOVL	REXP, @CEXP	1180
	28	AE	34	AE	3C	001D6	MOVZWL	R_DESC, R_LEN	1187
			24	AC	DD	001DB	PUSHL	CDIGITS	1188
			2C	AE	9F	001DE	PUSHAB	R_LEN	
			40	AE	DD	001E1	PUSHL	R_DESC+4	
	0000V	CF	03	FB	001E4		CALLS	#3, CHK_STR_TYPE	
			34	AE	9F	001E9	PUSHAB	R_DESC	1195
00000000G	00		01	FB	001EC		CALLS	#T, STR\$FREE1_DX	
			44	AE	9F	001F3	PUSHAB	A_DESC	1196
00000000G	00		01	FB	001F6		CALLS	#T, STR\$FREE1_DX	
			3C	AE	9F	001FD	PUSHAB	B_DESC	1197
00000000G	00		01	FB	00200		CALLS	#T, STR\$FREE1_DX	
			04	00207			RET		1198
			0000	00208	17\$:		.WORD	Save nothing	0968
		50	08	AC	D0	0020A	MOVL	8(AP), RO	
		50	04	A0	D0	0020E	MOVL	4(RO), RO	
			E8	A0	9F	00212	PUSHAB	R_DESC	
			F0	A0	9F	00215	PUSHAB	B_DESC	
			F8	A0	9F	00218	PUSHAB	A_DESC	
				03	DD	0021B	PUSHL	#3	
				5E	DD	0021D	PUSHL	SP	
		7E	04	AC	7D	0021F	MOVQ	4(AP), -(SP)	
	0000V	CF		03	FB	00223	CALLS	#3, FREE_STRINGS	
				04	00228		RET		

; Routine Size: 553 bytes, Routine Base: _STR\$CODE + 055C

STRARITH
1-019

M 12
16-Sep-1984 01:27:51
14-Sep-1984 12:40:01

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]STRARITH.B32;1

Page 31
(5)

: 1109

1199 1

```
1111 1200 1 GLOBAL ROUTINE STR$RECIP (
1112 1201 1     ASIGN,
1113 1202 1     AEXP,
1114 1203 1     ADIGITS,
1115 1204 1     BSIGN,
1116 1205 1     BEXP,
1117 1206 1     BDIGITS,
1118 1207 1     CSIGN,
1119 1208 1     CEXP,
1120 1209 1     CDIGITS
1121 1210 1 ) : NOVALUE =
1122 1211 1
1123 1212 1 ++
1124 1213 1 FUNCTIONAL DESCRIPTION:
1125 1214 1
1126 1215 1     Take the reciprocal of A, to precision B.  C := 1 / A
1127 1216 1
1128 1217 1 FORMAL PARAMETERS:
1129 1218 1
1130 1219 1     ASIGN.rv.l      0 = operand A is positive, 1 = negative
1131 1220 1     AEXP.rl.l      Power of 10 by which to multiply the operand A
1132 1221 1                  digits to get the absolute value of operand A.
1133 1222 1                  E.g., AEXP = 1, ADIGITS = 123 gives 1230.
1134 1223 1     ADIGITS.rnu.d   Descriptor for the digits of operand A
1135 1224 1     BSIGN.rv.l      0 = operand B is positive, 1 = negative
1136 1225 1     BEXP.rl.r      Power of 10 by which to multiply the operand B
1137 1226 1                  digits to get the absolute value of operand B.
1138 1227 1                  E.g., BEXP = -1, BDIGITS = 123 gives 12.3.
1139 1228 1     BDIGITS.rnu.d   Descriptor for the digits of operand B
1140 1229 1     CSIGN.wl.r      0 = operand C is positive, 1 = negative
1141 1230 1     CEXP.wl.r      Power of 10 by which to multiply the operand C
1142 1231 1                  digits to get the absolute value of operand C.
1143 1232 1                  E.g., CEXP = 0, CDIGITS = 123 gives 123.
1144 1233 1     CDIGITS.wnu.d   Descriptor for the digits of operand C
1145 1234 1
1146 1235 1 IMPLICIT INPUTS:
1147 1236 1
1148 1237 1     NONE
1149 1238 1
1150 1239 1 IMPLICIT OUTPUTS:
1151 1240 1
1152 1241 1     NONE
1153 1242 1
1154 1243 1 ROUTINE VALUE:
1155 1244 1 COMPLETION CODES:
1156 1245 1
1157 1246 1     NONE
1158 1247 1
1159 1248 1 SIDE EFFECTS:
1160 1249 1
1161 1250 1     May allocate space for the CDIGITS string.
1162 1251 1     Signals if memory is exhausted.
1163 1252 1     Signals Division by zero if operand A is zero.
1164 1253 1
1165 1254 1 --
1166 1255 1
1167 1256 2 BEGIN
```

```
1168 1257 2
1169 1258
1170 1259 MAP
1171 1260 ADIGITS : REF BLOCK [8, BYTE],
1172 1261 BDIGITS : REF BLOCK [8, BYTE],
1173 1262 CDIGITS : REF BLOCK [8, BYTE];
1174 1263
1175 1264 LOCAL
1176 1265 !+ Internal form of A.
1177 1266
1178 1267 A_DESC : BLOCK [8, BYTE] VOLATILE,
1179 1268 A_BUF : REF VECTOR [65535, BYTE],
1180 1269 A_LEN,
1181 1270 A_SIGN,
1182 1271 !+ Internal form of B.
1183 1272
1184 1273
1185 1274 B_DESC : BLOCK [8, BYTE] VOLATILE,
1186 1275 B_BUF : REF VECTOR [65535, BYTE],
1187 1276 B_LEN,
1188 1277 B_SIGN,
1189 1278 !+ The following are various auxiliary variables required to do the division
1190 1279 and check for its completion.
1191 1280
1192 1281
1193 1282 X_SIGN,
1194 1283 X_EXP,
1195 1284 X_DESC : BLOCK [8, BYTE] VOLATILE,
1196 1285 X_BUF : REF VECTOR [65535, BYTE],
1197 1286 X2_SIGN,
1198 1287 X2_EXP,
1199 1288 X2_DESC : BLOCK [8, BYTE] VOLATILE,
1200 1289 X2_BUF : REF VECTOR [65535, BYTE],
1201 1290 Q_SIGN,
1202 1291 Q_EXP,
1203 1292 Q_DESC : BLOCK [8, BYTE] VOLATILE,
1204 1293 Q_BUF : REF VECTOR [65535, BYTE],
1205 1294 Q_LEN,
1206 1295 XA_SIGN,
1207 1296 XA_EXP,
1208 1297 XA_DESC : BLOCK [8, BYTE] VOLATILE,
1209 1298 XA_BUF : REF VECTOR [65535, BYTE],
1210 1299 DELTA_SIGN,
1211 1300 DELTA_EXP,
1212 1301 DELTA_DESC : BLOCK [8, BYTE] VOLATILE,
1213 1302 DELTA_BUF : REF VECTOR [65535, BYTE],
1214 1303 ONE_DESC : BLOCK [8, BYTE],
1215 1304 ONE_BUF : VECTOR [1, BYTE],
1216 1305 ITER_DONE,
1217 1306 POS,
1218 1307
1219 1308 !+ The following are locals needed for calls to LIB$ANALYZE_SDESC.
1220 1309
1221 1310
1222 1311 CBUF,
1223 1312 C_LEN,
1224 1313 STATUS;
```

! Added for call to CHK_STR_TYPE

! 1 = the division process is done, exit its loop
! Power of ten by which we are dividing (shifting right)


```
1225 1314 2
1226 1315 BUILTIN
1227 1316 ACTUALCOUNT;
1228 1317
1229 1318 +
1230 1319 - Enable a handler to free the local strings in case of an error.
1231 1320
1232 1321
1233 1322 ENABLE
1234 1323 FREE_STRINGS (A_DESC, B_DESC, X_DESC, X2_DESC, Q_DESC, KA_DESC, DELTA_DESC);
1235 1324
1236 1325 +
1237 1326 - Check for the proper number of arguments.
1238 1327
1239 1328
1240 1329 IF (ACTUALCOUNT () LSS 9)
1241 1330 THEN
1242 1331 BEGIN
1243 1332
1244 1333 LOCAL
1245 1334 ROUT_NAME_DESC : BLOCK [8, BYTE];
1246 1335
1247 1336 ROUT_NAME_DESC [DSC$W_LENGTH] = 9;
1248 1337 ROUT_NAME_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_T;
1249 1338 ROUT_NAME_DESC [DSC$B_CLASS] = DSC$K_CLASS_S;
1250 1339 ROUT_NAME_DESC [DSC$A_POINTER] = UPLIT (ASCII 'STR$RECIP');
1251 1340 LIB$STOP (STR$_WRONUMARG, 2, ACTUALCOUNT (), ROUT_NAME_DESC);
1252 1341 END;
1253 1342
1254 1343 +
1255 1344 - Copy the A and B operands.
1256 1345
1257 1346 A_DESC [DSC$W_LENGTH] = 0;
1258 1347 A_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
1259 1348 A_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
1260 1349 A_DESC [DSC$A_POINTER] = 0;
1261 1350 +
1262 1351 - Compute the length of operand A. Only the leading digits count.
1263 1352 First call LIB$ANALYZE_SDESC to ensure that the input descriptor
1264 1353 is valid. If it is, then A_BUF will contain the address of the
1265 1354 first byte of the string, and A_LEN will contain its length.
1266 1355
1267 1356
1268 1357 STATUS = LIB$ANALYZE_SDESC (.ADIGITS, A_LEN, A_BUF);
1269 1358 IF .STATUS NEQ SS$_NORMAL
1270 1359 THEN
1271 1360 LIB$STOP (LIB$_INVARG);
1272 1361
1273 1362 +
1274 1363 - Also check here for the CDIGITS descriptor before getting too
1275 1364 involved in the routine.
1276 1365
1277 1366
1278 1367 STATUS = LIB$ANALYZE_SDESC (.CDIGITS, C_LEN, CBUF);
1279 1368 IF .STATUS NEQ SS$_NORMAL
1280 1369 THEN
1281 1370 LIB$STOP (LIB$_INVARG);
```

```
1282 1371 2
1283 1372 2
1284 1373 2
1285 1374 2
1286 1375 2
1287 1376 2
1288 1377 2
1289 1378 2
1290 1379 2
1291 1380 2
1292 1381 2
1293 1382 2
1294 1383 2
1295 1384 2
1296 1385 2
1297 1386 2
1298 1387 2
1299 1388 2
1300 1389 2
1301 1390 2
1302 1391 2
1303 1392 2
1304 1393 2
1305 1394 2
1306 1395 2
1307 1396 2
1308 1397 2
1309 1398 2
1310 1399 2
1311 1400 2
1312 1401 2
1313 1402 2
1314 1403 2
1315 1404 2
1316 1405 2
1317 1406 2
1318 1407 2
1319 1408 2
1320 1409 2
1321 1410 2
1322 1411 2
1323 1412 2
1324 1413 2
1325 1414 2
1326 1415 2
1327 1416 2
1328 1417 2
1329 1418 2
1330 1419 2
1331 1420 2
1332 1421 2
1333 1422 2
1334 1423 2
1335 1424 2
1336 1425 2
1337 1426 2
1338 1427 2

A_LEN = 0;
A_SIGN = ..ASIGN;
BEGIN

LOCAL
    SCAN_DONE;

SCAN_DONE = 0;

DO
    BEGIN
        IF (.A_LEN EQLU .ADIGITS [DSC$W_LENGTH])
        THEN
            SCAN_DONE = 1
        ELSE
            IF ((.A_BUF [.A_LEN] GEQ %C'0') AND (.A_BUF [.A_LEN] LEQ %C'9'))
            THEN
                A_LEN = .A_LEN + 1
            ELSE
                SCAN_DONE = 1;

        END
    UNTIL (.SCAN_DONE);

END;
STR$GET1_DX (A_LEN, A_DESC);
A_BUF = .A_DESC [DSC$A_POINTER];
CH$MOVE (.A_LEN, .ADIGITS [DSC$A_POINTER], A_BUF [0]);

+
- If operand A is zero, fail.

IF CH$EQL (1, CH$PTR (UPLIT ('0')), .A_LEN, A_BUF [0], %C'0') THEN LIB$STOP (STR$_DIVBY_ZER);

B_DESC [DSC$W_LENGTH] = 0;
B_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
B_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
B_DESC [DSC$A_POINTER] = 0;

+
- Compute the length of operand B. Only the leading digits count.
First call LIB$ANALYZE_SDESC to ensure that the input descriptor
is valid. If it is, then B_BUF will contain the address of the
first byte of the string, and B_LEN will contain its length.

STATUS = LIB$ANALYZE_SDESC (.BDIGITS, B_LEN, B_BUF);
IF .STATUS NEQ $$$_NORMAL
THEN
    LIB$STOP (LIB$_INVARG);
B_LEN = 0;
B_SIGN = ..BSIGN;
BEGIN

LOCAL
```

```
1339 1428 SCAN_DONE;  
1340 1429  
1341 1430 SCAN_DONE = 0;  
1342 1431  
1343 1432 DO  
1344 1433 BEGIN  
1345 1434  
1346 1435 IF (.B_LEN EQLU .BDIGITS [DSC$W_LENGTH])  
1347 1436 THEN  
1348 1437 SCAN_DONE = 1  
1349 1438 ELSE  
1350 1439  
1351 1440 IF ((.B_BUF [.B_LEN] GEQ %C'0') AND (.B_BUF [.B_LEN] LEQ %C'9'))  
1352 1441 THEN  
1353 1442 B_LEN = .B_LEN + 1  
1354 1443 ELSE  
1355 1444 SCAN_DONE = 1;  
1356 1445  
1357 1446 END  
1358 1447 UNTIL (.SCAN_DONE);  
1359 1448  
1360 1449 END;  
1361 1450 STR$GET1_DX (B_LEN, B_DESC);  
1362 1451 B_BUF = .B_DESC [DSC$A_POINTER];  
1363 1452 CR$MOVE (.B_LEN, .BDIGITS [DSC$A_POINTER], B_BUF [0]);  
1364 1453  
1365 1454 + Initialize the auxiliary variables.  
1366 1455 -  
1367 1456 X_DESC [DSC$W_LENGTH] = 0;  
1368 1457 X_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;  
1369 1458 X_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;  
1370 1459 X_DESC [DSC$A_POINTER] = 0;  
1371 1460 STR$GET1_DX (%REF (1), X_DESC);  
1372 1461 X_BUF = .X_DESC [DSC$A_POINTER];  
1373 1462 X_BUF [0] = %C'1';  
1374 1463 X_SIGN = 0;  
1375 1464 X_EXP = 0;  
1376 1465  
1377 1466 X2_DESC [DSC$W_LENGTH] = 0;  
1378 1467 X2_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;  
1379 1468 X2_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;  
1380 1469 X2_DESC [DSC$A_POINTER] = 0;  
1381 1470 STR$GET1_DX (%REF (1), X2_DESC);  
1382 1471 X2_BUF = .X2_DESC [DSC$A_POINTER];  
1383 1472 X2_BUF [0] = %C'0';  
1384 1473 X2_SIGN = 0;  
1385 1474 X2_EXP = 0;  
1386 1475  
1387 1476 Q_DESC [DSC$W_LENGTH] = 0;  
1388 1477 Q_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;  
1389 1478 Q_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;  
1390 1479 Q_DESC [DSC$A_POINTER] = 0;  
1391 1480 STR$GET1_DX (%REF (1), Q_DESC);  
1392 1481 Q_BUF = .Q_DESC [DSC$A_POINTER];  
1393 1482 Q_BUF [0] = %C'0';  
1394 1483 Q_SIGN = 0;  
1395 1484 Q_EXP = 0;
```

```
1396 1485 2 !
1397 1486 2 !
1398 1487 2 !
1399 1488 2 !
1400 1489 2 !
1401 1490 2 !
1402 1491 2 !
1403 1492 2 !
1404 1493 2 !
1405 1494 2 !
1406 1495 2 !
1407 1496 2 !
1408 1497 2 !
1409 1498 2 !
1410 1499 2 !
1411 1500 2 !
1412 1501 2 !
1413 1502 2 !
1414 1503 2 !
1415 1504 2 !
1416 1505 2 !
1417 1506 2 !
1418 1507 2 !
1419 1508 2 !
1420 1509 2 !
1421 1510 2 !
1422 1511 2 !
1423 1512 2 !
1424 1513 2 !
1425 1514 2 !
1426 1515 2 !
1427 1516 2 !
1428 1517 2 !
1429 1518 2 !
1430 1519 2 !
1431 1520 2 !
1432 1521 2 !
1433 1522 2 !
1434 1523 2 !
1435 1524 2 !
1436 1525 2 !
1437 1526 2 !
1438 1527 2 !
1439 1528 2 !
1440 1529 2 !
1441 1530 2 !
1442 1531 2 !
1443 1532 2 !
1444 1533 2 !
1445 1534 2 !
1446 1535 2 !
1447 1536 2 !
1448 1537 2 !
1449 1538 2 !
1450 1539 2 !
1451 1540 2 !
1452 1541 2 !

XA_DESC [DSC$W_LENGTH] = 0;
XA_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
XA_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
XA_DESC [DSC$A_POINTER] = 0;
STR$GET1_DX (XREF (1), XA_DESC);
XA_BUF = XA_DESC [DSC$A_POINTER];
XA_BUF [0] = '0';
XA_SIGN = 0;
XA_EXP = 0;

DELTA_DESC [DSC$W_LENGTH] = 0;
DELTA_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
DELTA_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
DELTA_DESC [DSC$A_POINTER] = 0;
STR$GET1_DX (XREF (1), DELTA_DESC);
DELTA_BUF = DELTA_DESC [DSC$A_POINTER];
DELTA_BUF [0] = '0';
DELTA_SIGN = 0;
DELTA_EXP = 0;

ONE_DESC [DSC$W_LENGTH] = 1;
ONE_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
ONE_DESC [DSC$B_CLASS] = DSC$K_CLASS_S;
ONE_DESC [DSC$A_POINTER] = ONE_BUF;
ONE_BUF [0] = '1';

+ Decide on the best position to start forming the quotient. Unless
the divisor is 1, the first subtract will cause X to go negative
and force us to back off.
- POS = -.X_EXP;

+ Iterate until we are close to the quotient.
If B = 0, this will take a long time.
- ITER_DONE = 0;
DO
  BEGIN
    STR$ADD (X_SIGN, X_EXP, X_DESC,
             XREF (T), XREF (..AEXP + .POS), A_DESC,
             X_SIGN, X_EXP, X_DESC);
    + If we have gone negative, back off. Otherwise increase the quotient.
    - IF (.X_SIGN)
      THEN
        BEGIN
          STR$ADD (X_SIGN, X_EXP, X_DESC,
                   XREF (0), XREF (..AEXP + .POS), A_DESC,
                   X_SIGN, X_EXP, X_DESC);
          + Go down to the next lower digit
          - POS = .POS - 1;
```



```
1453 1542 4 1+
1454 1543 4 1+ Now see if we are close enough to the reciprocal.
1455 1544 4 1-
1456 1545 4 STRSMUL (Q SIGN, Q EXP, Q DESC, !
1457 1546 4 XREF (0), .AEXP, A DESC, !
1458 1547 4 XA SIGN, XA EXP, XA DESC);
1459 1548 4 STRSADD (XA SIGN, XA EXP, XA DESC, !
1460 1549 4 XREF (1), XREF (0), ONE DESC, !
1461 1550 4 DELTA SIGN, DELTA EXP, DELTA DESC);
1462 1551 4 DELTA SIGN = 0;
1463 1552 4 STRSADD (DELTA SIGN, DELTA EXP, DELTA DESC, !
1464 1553 4 XREF (1), .BEXP, B DESC, !
1465 1554 4 X2 SIGN, X2 EXP, X2 DESC);
1466 1555 4
1467 1556 5 IF (.X2_SIGN)
1468 1557 4 THEN
1469 1558 4 ITER_DONE = 1
1470 1559 4 ELSE
1471 1560 4
1472 1561 5 IF (.DELTA_DESC [DSC$W_LENGTH] EQLU 1)
1473 1562 4 THEN
1474 1563 3 BEGIN
1475 1564 3 LOCAL
1476 1565 3 DELTA_BUF : REF VECTOR [65535, BYTE];
1477 1566 3
1478 1567 3 DELTA_BUF = .DELTA_DESC [DSC$A_POINTER];
1479 1568 3
1480 1569 3 IF (.DELTA_BUF [0] EQL %C'O') THEN ITER_DONE = 1;
1481 1570 3
1482 1571 3 END;
1483 1572 4
1484 1573 4
1485 1574 4
1486 1575 3 END
1487 1576 4 ELSE
1488 1577 4 BEGIN
1489 1578 4 STRSADD (Q SIGN, Q EXP, Q DESC, !
1490 1579 4 XREF (0), POS, ONE DESC, !
1491 1580 4 Q_SIGN, Q_EXP, Q_DESC);
1492 1581 4
1493 1582 4 END;
1494 1583 2 UNTIL (.ITER_DONE);
1495 1584 2
1496 1585 2 1+
1497 1586 2 The reciprocal now lives in Q. Return it to the caller with the
1498 1587 2 original sign of A, which was not used above.
1499 1588 2 1-
1500 1589 2 .CSIGN = .A SIGN;
1501 1590 2 .CEXP = .Q_EXP;
1502 1591 2
1503 1592 2 1+
1504 1593 2 Call CHK_STR_TYPE to determine if we need to pad the number with
1505 1594 2 leading zeroes depending on the string type.
1506 1595 2 1-
1507 1596 2
1508 1597 2 QLEN = .Q_DESC[DSC$W_LENGTH];
1509 1598 2 CHK_STR_TYPE (.Q_DESC[DSC$A_POINTER], QLEN, .CDIGITS);
```

```
1510 1599 2
1511 1600 2
1512 1601 2
1513 1602 2
1514 1603 2
1515 1604 2
1516 1605 2
1517 1606 2
1518 1607 2
1519 1608 2
1520 1609 2
1521 1610 1
```

STR\$COPY_DX (.CDIGITS, Q_DESC);
Free our strings.
STR\$FREE1_DX (X_DESC);
STR\$FREE1_DX (X2_DESC);
STR\$FREE1_DX (Q_DESC);
STR\$FREE1_DX (XA_DESC);
STR\$FREE1_DX (DELTA_DESC);
END;

! end of STR\$RECIP

```
00 00 00 50 49 43 45 52 24 52 54 53 00785
00 00 00 00 00 00 30 00788 P.AAG: .BLKB 3
00794 P.AAH: .ASCII \STR$RECIP\<0><0><0>
.ASCII \0\<0><0><0>
```

```
OFFC 00000 .ENTRY STR$RECIP, Save R2,R3,R4,R5,R6,R7,R8,R9,-
5B F97E CF 9E 00002 MOVAB STR$ADD, R11
5A 00000000G 00 9E 00007 MOVAB STR$FREE1_DX, R10
59 00000000G 00 9E 0000E MOVAB LIB$STOP, R9
58 00000000G 00 9E 00015 MOVAB STR$GET1_DX, R8
5E FF64 CE 9E 0001C MOVAB -156(SP), SP
64 AE 7C 00021 CLRQ DELTA_DESC
6C AE 7C 00024 CLRQ XA_DESC
74 AE 7C 00027 CLRQ Q_DESC
7C AE 7C 0002A CLRQ X2_DESC
E8 AD 7C 0002D CLRQ X_DESC
F0 AD 7C 00030 CLRQ B_DESC
F8 AD 7C 00033 CLRQ A_DESC
6D 0353 CF DE 00036 MOVAL 18$, (FP)
09 6C 91 0003B CMPB (AP), #9
54 AE 010E0009 1E 1E 0003E BGEQU 1$
58 AE A5 AF 9E 00040 MOVL #17694729, ROUT_NAME_DESC
AE 54 AE 9F 00048 MOVAB P.AAG, ROUT_NAME_DESC+4
7E 6C 9A 00050 PUSHAB ROUT_NAME_DESC
02 DD 00053 MOVZBL (AP), -(SP)
04 FB 0005B PUSHL #2
69 00000000G 8F DD 00055 PUSHL #STR$WRONUMARG
FB AD B4 0005E CALLS #4, LIB$STOP
FA AD 0F 90 00061 CLRW A_DESC
FB AD 02 90 00065 MOVAB #15, A_DESC+2
FC AD D4 00069 MOVAB #2, A_DESC+3
08 AE 9F 0006C CLRL A_DESC+4
18 AE 9F 0006F PUSHAB A_BUF
52 0C AC D0 00072 PUSHAB A_LEN
00000000G 00 03 FB 00078 MOVL ADIGITS, R2
56 50 D0 0007F PUSHL R2
01 56 D1 00082 CALLS #3, LIB$ANALYZE_SDESC
09 13 00085 MOVL R0, STATUS
BEQL STATUS, #1
```

				00000000G	8F	DD	00087	PUSHL	#LIB\$ INVARG	1360
				69	01	FB	0008D	CALLS	#1, LIB\$STOP	
					0C	AE	9F	00090	2\$: PUSHAB	1367
					14	AE	9F	00093	CBUF	
					24	AC	DD	00096	C_LEN	
				00000000G	00	03	FB	00099	PUSHL	
				56	50	DD	000A0	CALLS	#3, LIB\$ANALYZE_SDESC	
				01	56	D1	000A3	MOVL	R0, STATUS	
					09	13	000A6	CMPL	STATUS, #1	1368
					00000000G	8F	DD	000A8	BEQL	3\$
				69	01	FB	000AE	PUSHL	#LIB\$ INVARG	1370
					14	AE	D4	000B1	CALLS	#1, LIB\$STOP
				57	04	BC	DD	000B4	3\$: CLRL	1372
					51	D4	000B8	MOVL	@ASIGN, A_SIGN	1373
14	AE	62		10	00	ED	000BA	CLRL	SCAN DONE	1379
		50	08	AE	15	13	000C0	4\$: CMPZV	#0, #16, (R2), A_LEN	1384
				30	14	AE	C1	000C2	BEQL	5\$
					60	91	000C8	ADDL3	A_LEN, A_BUF, R0	1389
				39	0A	1F	000CB	CMPB	(R0), #48	
					60	91	000CD	BLSSU	5\$	
					05	1A	000D0	CMPB	(R0), #57	
					14	AE	D6	000D2	BGTRU	5\$
					03	11	000D5	INCL	A_LEN	1391
				51	01	DD	000D7	BRB	6\$	
				DD	51	E9	000DA	5\$: MOVL	#1, SCAN DONE	1393
					F8	AD	9F	000DD	6\$: BLBC	SCAN DONE, 4\$
					18	AE	9F	000E0	PUSHAB	A_DESC
					02	FB	000E3	PUSHAB	A_LEN	1399
				68	02	FB	000E3	CALLS	#2, STR\$GET1 DX	
				AE	FC	AD	DD	000E6	MOVL	A_DESC+4, A_BUF
14	AE	08	BE	08	14	AE	28	000EB	MOV3	A_LEN, @4(R2), @A_BUF
			30	04	01	2D	000F2	CMPC5	#T, P.AAH, #48, A_LEN, @A_BUF	1400
				FF05	08	BE	000FA			1401
					09	12	000FC			1406
					00000000G	8F	DD	000FE	BNEQ	7\$
				69	01	FB	00104	PUSHL	#STR\$ DIVBY_ZER	
					F0	AD	B4	00107	CALLS	#1, LIB\$STOP
				F2	AD	0F	90	0010A	7\$: CLRW	B_DESC
				F3	AD	02	90	0010E	MOVB	#T5, B_DESC+2
					F4	AD	D4	00112	MOVB	#2, B_DESC+3
					18	AE	9F	00115	CLRL	B_DESC+4
					20	AE	9F	00118	PUSHAB	B_BUF
				52	18	AC	DD	0011B	PUSHAB	B_LEN
					52	DD	0011F	MOVL	B_LEN	
					00000000G	03	FB	00121	PUSHL	R2
				56	50	DD	00128	CALLS	#3, LIB\$ANALYZE_SDESC	
				01	56	D1	00128	MOVL	R0, STATUS	
					09	13	0012E	CMPL	STATUS, #1	1420
					00000000G	8F	DD	00130	BEQL	8\$
				69	01	FB	00136	PUSHL	#LIB\$ INVARG	1422
					1C	AE	D4	00139	CALLS	#1, LIB\$STOP
				50	10	BC	DD	0013C	8\$: CLRL	B_LEN
					51	D4	00140	MOVL	@BSIGN, B_SIGN	1423
				10	00	ED	00142	CLRL	SCAN DONE	1424
1C	AE	62			15	13	00148	9\$: CMPZV	#0, #16, (R2), B_LEN	1430
		50	18	AE	1C	AE	C1	0014A	BEQL	10\$
				30	60	91	00150	ADDL3	B_LEN, B_BUF, R0	1440
					0A	1F	00153	CMPB	(R0), #48	
								BLSSU	10\$	

		39		60	91	00155	CMPB	(R0), #57		
				05	1A	00158	BGTRU	108		
			1C	AE	D6	0015A	INCL	B_LEN	1442	
				03	11	0015D	BRB	1T\$		
		51		01	D0	0015F	MOVL	#1, SCAN_DONE	1444	
		DD		51	E9	00162	BLBC	SCAN_DONE, 9\$	1447	
			F0	AD	9F	00165	PUSHAB	B_DESC	1450	
			20	AE	9F	00168	PUSHAB	B_LEN		
		68		02	FB	0016B	CALLS	#2, STR\$GET1_DX		
		AE	F4	AD	D0	0016E	MOVL	B_DESC+4, B_BUF	1451	
18	BE	04	B2	1C	AE	28	MOVC3	B_LEN, @4(R2), @B_BUF	1452	
			E8	AD	B4	0017A	CLRW	X_DESC	1456	
		EA	AD	0F	90	0017D	MOVB	#15, X_DESC+2	1457	
		EB	AD	02	90	00181	MOVB	#2, X_DESC+3	1458	
			EC	AD	D4	00185	CLRL	X_DESC+4	1459	
			E8	AD	9F	00188	PUSHAB	X_DESC	1460	
		08	AE	01	D0	0018B	MOVL	#1, 8(SP)		
			08	AE	9F	0018F	PUSHAB	8(SP)		
		68		02	FB	00192	CALLS	#2, STR\$GET1_DX		
		50		EC	AD	D0	MOVL	X_DESC+4, X_BUF	1461	
		60		31	90	00199	MOVB	#29, (X_BUF)	1462	
			24	AE	7C	0019C	CLRQ	X_EXP	1464	
			7C	AE	B4	0019F	CLRW	X2_DESC	1466	
		7E	AE	0F	90	001A2	MOVB	#15, X2_DESC+2	1467	
		7F	AE	02	90	001A6	MOVB	#2, X2_DESC+3	1468	
			E4	AD	D4	001AA	CLRL	X2_DESC+4	1469	
			7C	AE	9F	001AD	PUSHAB	X2_DESC	1470	
		08	AE	01	D0	001B0	MOVL	#1, 8(SP)		
			08	AE	9F	001B4	PUSHAB	8(SP)		
		68		02	FB	001B7	CALLS	#2, STR\$GET1_DX		
		50		EC	AD	D0	MOVL	X2_DESC+4, X2_BUF	1471	
		60		30	90	001BE	MOVB	#48, (X2_BUF)	1472	
			34	AE	7C	001C1	CLRQ	X2_EXP	1474	
			74	AE	B4	001C4	CLRW	Q_DESC	1476	
		76	AE	0F	90	001C7	MOVB	#15, Q_DESC+2	1477	
		77	AE	02	90	001CB	MOVB	#2, Q_DESC+3	1478	
			78	AE	D4	001CF	CLRL	Q_DESC+4	1479	
			74	AE	9F	001D2	PUSHAB	Q_DESC	1480	
		08	AE	01	D0	001D5	MOVL	#1, 8(SP)		
			08	AE	9F	001D9	PUSHAB	8(SP)		
		68		02	FB	001DC	CALLS	#2, STR\$GET1_DX		
		50		78	AE	D0	MOVL	Q_DESC+4, Q_BUF	1481	
		60		30	90	001E3	MOVB	#48, (Q_BUF)	1482	
			48	AE	7C	001E6	CLRQ	Q_EXP	1484	
			6C	AE	B4	001E9	CLRW	XA_DESC	1486	
		6E	AE	0F	90	001EC	MOVB	#15, XA_DESC+2	1487	
		6F	AE	02	90	001F0	MOVB	#2, XA_DESC+3	1488	
			70	AE	D4	001F4	CLRL	XA_DESC+4	1489	
			6C	AE	9F	001F7	PUSHAB	XA_DESC	1490	
		08	AE	01	D0	001FA	MOVL	#1, 8(SP)		
			08	AE	9F	001FE	PUSHAB	8(SP)		
		68		02	FB	00201	CALLS	#2, STR\$GET1_DX		
		50		70	AE	D0	MOVL	XA_DESC+4, XA_BUF	1491	
		60		30	90	00208	MOVB	#48, (XA_BUF)	1492	
			2C	AE	7C	0020B	CLRQ	XA_EXP	1494	
			64	AE	B4	0020E	CLRW	DELTA_DESC	1496	
		66	AE	0F	90	00211	MOVB	#15, DELTA_DESC+2	1497	

67	AE		02	90	00215	MOVB	#2, DELTA_DESC+3	1498	
		68	AE	D4	00219	CLRL	DELTA_DESC+4	1499	
		64	AE	9F	0021C	PUSHAB	DELTA_DESC	1500	
08	AE		01	D0	0021F	MOVL	#1, 8(SP)		
		08	AE	9F	00223	PUSHAB	8(SP)		
			02	FB	00226	CALLS	#2, STR\$GET1_DX		
		68	AE	D0	00229	MOVL	DELTA_DESC+4, DELTA_BUF	1501	
		50							
		60	30	90	0022D	MOVB	#48, TDELTA_BUF)	1502	
			3C	AE	7C	00230	CLRL	DELTA_EXP	1504
5C	AE	010FD001	8F	D0	00233	MOVL	#17760257, ONE_DESC	1506	
60	AE		20	AE	9E	0023B	MOVAB	ONE_BUF, ONE_DESC+4	1509
20	AE		31	90	00240	MOVB	#49, ONE_BUF	1510	
44	AE		24	AE	CE	00244	MNEGL	X_EXP, POS	1516
			53	D4	00249	CLRL	ITER_DONE	1521	
			E8	AD	9F	0024B	PUSHAB	X_DESC	1525
			28	AE	9F	0024E	PUSHAB	X_EXP	
			30	AE	9F	00251	PUSHAB	X_SIGN	
			F8	AD	9F	00254	PUSHAB	A_DESC	
52	08	BC	54	AE	C1	00257	ADDL3	POS, @AEXP, R2	1526
	14	AE		52	D0	0025D	MOVL	R2, 20(SP)	
			14	AE	9F	00261	PUSHAB	20(SP)	
	14	AE		01	D0	00264	MOVL	#1, 20(SP)	
			14	AE	9F	00268	PUSHAB	20(SP)	
			E8	AD	9F	0026B	PUSHAB	X_DESC	1525
			40	AE	9F	0026E	PUSHAB	X_EXP	
			48	AE	9F	00271	PUSHAB	X_SIGN	
	6B		09	FB	00274	CALLS	#9, STR\$ADD		
	03		28	AE	E8	00277	BLBS	X_SIGN, 13\$	1532
				00AD	31	0027B	BRW	15\$	
			E8	AD	9F	0027E	PUSHAB	X_DESC	1535
			28	AE	9F	00281	PUSHAB	X_EXP	
			30	AE	9F	00284	PUSHAB	X_SIGN	
			F8	AD	9F	00287	PUSHAB	A_DESC	
	14	AE		52	D0	0028A	MOVL	R2, 20(SP)	1536
			14	AE	9F	0028E	PUSHAB	20(SP)	
			14	AE	D4	00291	CLRL	20(SP)	
			14	AE	9F	00294	PUSHAB	20(SP)	
			E8	AD	9F	00297	PUSHAB	X_DESC	1535
			40	AE	9F	0029A	PUSHAB	X_EXP	
			48	AE	9F	0029D	PUSHAB	X_SIGN	
	6B		09	FB	002A0	CALLS	#9, STR\$ADD		
			44	AE	D7	002A3	DECL	POS	1541
			6C	AE	9F	002A6	PUSHAB	XA_DESC	1545
			30	AE	9F	002A9	PUSHAB	XA_EXP	
			38	AE	9F	002AC	PUSHAB	XA_SIGN	
			F8	AD	9F	002AF	PUSHAB	A_DESC	
			08	AC	DD	002B2	PUSHL	AEXP	1546
			18	AE	D4	002B5	CLRL	24(SP)	
			18	AE	9F	002B8	PUSHAB	24(SP)	
			D8	AD	9F	002BB	PUSHAB	Q_DESC	1545
			64	AE	9F	002BE	PUSHAB	Q_EXP	
			6C	AE	9F	002C1	PUSHAB	Q_SIGN	
0440	CB		09	FB	002C4	CALLS	#9, STR\$MUL		
			64	AE	9F	002C9	PUSHAB	DELTA_DESC	1548
			40	AE	9F	002CC	PUSHAB	DELTA_EXP	
			48	AE	9F	002CF	PUSHAB	DELTA_SIGN	
			68	AE	9F	002D2	PUSHAB	ONE_DESC	

		14	AE	D4	002D5	CLRL	20(SP)	1549
		14	AE	9F	002D8	PUSHAB	20(SP)	
14	AE		01	D0	002DB	MOVL	#1, 20(SP)	
		14	AE	9F	002DF	PUSHAB	20(SP)	
		D0	AD	9F	0C2E2	PUSHAB	XA_DESC	1548
		48	AE	9F	0C2E5	PUSHAB	XA_EXP	
		50	AE	9F	0C2E8	PUSHAB	XA_SIGN	
	6B		09	FB	002EB	CALLS	#9, STR\$ADD	
		40	AE	D4	002EE	CLRL	DELTA_SIGN	1551
		7C	AE	9F	002F1	PUSHAB	X2_DESC	1552
		38	AE	9F	002F4	PUSHAB	X2_EXP	
		40	AE	9F	002F7	PUSHAB	X2_SIGN	
		F0	AD	9F	002FA	PUSHAB	B_DESC	
		14	AC	DD	002FD	PUSHL	BEXP	1553
18	AE		01	D0	00300	MOVL	#1, 24(SP)	
		18	AE	9F	00304	PUSHAB	24(SP)	
		7C	AE	9F	00307	PUSHAB	DELTA_DESC	1552
		58	AE	9F	0030A	PUSHAB	DELTA_EXP	
		60	AE	9F	0030D	PUSHAB	DELTA_SIGN	
	6B		09	FB	00310	CALLS	#9, STR\$ADD	
	0F	38	AE	E8	00313	BLBS	X2_SIGN, 14\$	1556
	01	64	AE	B1	00317	CMPW	DELTA_DESC, #1	1561
			2F	12	0031B	BNEQ	16\$	
	50	68	AE	D0	0031D	MOVL	DELTA_DESC+4, DELTA_BUF	1568
	30		60	91	00321	CMPB	(DELTA_BUF), #48	1570
			26	12	00324	BNEQ	16\$	
	53		01	D0	00326	MOVL	#1, ITER_DONE	
			21	11	00329	BRB	16\$	1532
		74	AE	9F	0032B	PUSHAB	Q_DESC	1577
		4C	AE	9F	0032E	PUSHAB	Q_EXP	
		54	AE	9F	00331	PUSHAB	Q_SIGN	
		68	AE	9F	00334	PUSHAB	ONE_DESC	
		54	AE	9F	00337	PUSHAB	POS	
		18	AE	D4	0033A	CLRL	24(SP)	1578
		18	AE	9F	0033D	PUSHAB	24(SP)	
		D8	AD	9F	00340	PUSHAB	Q_DESC	1577
		64	AE	9F	00343	PUSHAB	Q_EXP	
		6C	AE	9F	00346	PUSHAB	Q_SIGN	
	6B		09	FB	00349	CALLS	#9, STR\$ADD	
	03		53	E8	0034C	BLBS	ITER_DONE, 17\$	1583
			FEF9	31	0034F	BRW	12\$	
			57	D0	00352	MOVL	A_SIGN, @CSIGN	1589
1C	BC					MOVL	Q_EXP, @CEXP	1590
20	BC	48	AE	D0	00356	MOVZWL	Q_DESC, QLEN	1597
50	AE	74	AE	3C	0035B	PUSHL	CDIGITS	1598
		24	AC	DD	00360	PUSHAB	QLEN	
		54	AE	9F	00363	PUSHL	Q_DESC+4	
		DC	AD	DD	00366	CALLS	#3, CHK_STR_TYPE	
0000V	CF		03	FB	00369	PUSHAB	X_DESC	1605
		E8	AD	9F	0036E	CALLS	#T, STR\$FREE1_DX	
	6A		01	FB	00371	PUSHAB	X2_DESC	1606
		7C	AE	9F	00374	CALLS	#1, STR\$FREE1_DX	
	6A		01	FB	00377	PUSHAB	Q_DESC	1607
		74	AE	9F	0037A	CALLS	#T, STR\$FREE1_DX	
	6A		01	FB	0037D	PUSHAB	XA_DESC	1608
		6C	AE	9F	00380	CALLS	#1, STR\$FREE1_DX	
	6A		01	FB	00383	PUSHAB	DELTA_DESC	1609
		64	AE	9F	00386			

6A		01	FB	00389	CALLS	#1, STR\$FREE1_DX
			04	0038C	RET	
		0000		0038D	18\$:	
50	08	AC	DD	0038F	.WORD	Save nothing
50	04	A0	DD	00393	MOVL	8(AP), R0
	C8	A0	9F	00397	MOVL	4(R0), R0
	D0	A0	9F	0039A	PUSHAB	DELTA_DESC
	D8	A0	9F	0039D	PUSHAB	XA_DESC
	E0	A0	9F	003A0	PUSHAB	Q_DESC
	E8	A0	9F	003A3	PUSHAB	X2_DESC
	F0	A0	9F	003A6	PUSHAB	X_DESC
	F8	A0	9F	003A9	PUSHAB	B_DESC
		07	DD	003AC	PUSHAB	A_DESC
		5E	DD	003AE	PUSHL	#7
	7E	AC	7D	003B0	PUSHL	SP
0000V	CF	03	FB	003B4	MOVQ	4(AP), -(SP)
			04	003B9	CALLS	#3, FREE_STRINGS
					RET	

.....1610
.....1256
.....

; Routine Size: 954 bytes. Routine Base: _STR\$CODE + 0798

; 1522 1611 1

```
1524 1612 1 GLOBAL ROUTINE STR$ROUND (
1525 1613 1     PLACES,
1526 1614 1     TRUNC,
1527 1615 1     ASIGN,
1528 1616 1     AEXP,
1529 1617 1     ADIGITS,
1530 1618 1     BSIGN,
1531 1619 1     BEXP,
1532 1620 1     BDIGITS
1533 1621 1 ) : NOVALUE =
1534 1622 1
1535 1623 1 **
1536 1624 1 FUNCTIONAL DESCRIPTION:
1537 1625 1
1538 1626 1     Round or truncate a number to a specified number of significant
1539 1627 1     digits.  B := ROUND (A)
1540 1628 1
1541 1629 1 FORMAL PARAMETERS:
1542 1630 1
1543 1631 1     PLACES.rl.r      Max decimal digits to retain in the result
1544 1632 1     TRUNC.rv.r      0 = round, 1 = truncate to that many places
1545 1633 1     ASIGN.rv.r      0 = operand A is positive, 1 = negative
1546 1634 1     AEXP.rl.r      Power of 10 by which to multiply the operand A
1547 1635 1                  digits to get the absolute value of operand A.
1548 1636 1                  E.g., AEXP = 1, ADIGITS = 123 gives 1230.
1549 1637 1     ADIGITS.rnu.d    Descriptor for the digits of operand A
1550 1638 1     BSIGN.wl.r      0 = operand B is positive, 1 = negative
1551 1639 1     BEXP.wl.r      Power of 10 by which to multiply the operand B
1552 1640 1                  digits to get the absolute value of operand B.
1553 1641 1                  E.g., BEXP = -1, BDIGITS = 123 gives 12.3.
1554 1642 1     BDIGITS.wnu.d    Descriptor for the digits of operand B
1555 1643 1
1556 1644 1 IMPLICIT INPUTS:
1557 1645 1
1558 1646 1     NONE
1559 1647 1
1560 1648 1 IMPLICIT OUTPUTS:
1561 1649 1
1562 1650 1     NONE
1563 1651 1
1564 1652 1 ROUTINE VALUE:
1565 1653 1 COMPLETION CODES:
1566 1654 1
1567 1655 1     NONE
1568 1656 1
1569 1657 1 SIDE EFFECTS:
1570 1658 1
1571 1659 1     May allocate space for the BDIGITS string.
1572 1660 1     Signals if memory is exhausted.
1573 1661 1
1574 1662 1 --
1575 1663 1
1576 1664 2 BEGIN
1577 1665 2
1578 1666 2 MAP
1579 1667 2     ADIGITS : REF BLOCK [8, BYTE],
1580 1668 2     BDIGITS : REF BLOCK [8, BYTE];
```



```
1581 1669 LOCAL
1582 1670
1583 1671 + Internal form of A.
1584 1672 -
1585 1673 ABUF : REF VECTOR [65535, BYTE],
1586 1674 A_SIGN,
1587 1675 + Internal form of B.
1588 1676 -
1589 1677 REXP,
1590 1678 R_DESC : BLOCK [8, BYTE] VOLATILE,
1591 1679 RBUF : REF VECTOR [65535, BYTE],
1592 1680 R_LEN, ! Length of the result
1593 1681 RESULT_DIGITS, ! Number of digits in the result
1594 1682
1595 1683
1596 1684 + The following locals are needed for calls to LIB$ANALYZE_SDESC.
1597 1685 -
1598 1686 A_LEN,
1599 1687 B_LEN,
1600 1688 BBUF,
1601 1689 STATUS;
1602 1690
1603 1691 BUILTIN
1604 1692 ACTUALCOUNT;
1605 1693
1606 1694 + Enable a handler to free the local string in case of an error.
1607 1695 -
1608 1696
1609 1697 ENABLE
1610 1698 FREE_STRINGS (R_DESC);
1611 1699
1612 1700 + Check for the proper number of arguments.
1613 1701 -
1614 1702
1615 1703 IF (ACTUALCOUNT () LSS B)
1616 1704 THEN
1617 1705 BEGIN
1618 1706 LOCAL
1619 1707 ROUT_NAME_DESC : BLOCK [8, BYTE];
1620 1708
1621 1709 ROUT_NAME_DESC [DSC$W_LENGTH] = 9;
1622 1710 ROUT_NAME_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_T;
1623 1711 ROUT_NAME_DESC [DSC$B_CLASS] = DSC$K_CLASS_S;
1624 1712 ROUT_NAME_DESC [DSC$A_POINTER] = UPLIT (%ASCII'STR$ROUND');
1625 1713 LIB$STOP (STR$WRONUMARG, 2, ACTUALCOUNT (), ROUT_NAME_DESC);
1626 1714 END;
1627 1715
1628 1716 + Copy the given number to local storage before we begin work on it.
1629 1717 -
1630 1718 A_SIGN = ..ASIGN;
1631 1719 REXP = ..AREXP;
```

```
1638 1726 2 R_DESC [DSC$W_LENGTH] = 0;
1639 1727 2 R_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_NU;
1640 1728 2 R_DESC [DSC$B_CLASS] = DSC$K_CLASS_D;
1641 1729 2 R_DESC [DSC$A_POINTER] = 0;
1642 1730 2
1643 1731 2 + Compute the length of operand A. Only the leading digits count.
1644 1732 2 First call LIB$ANALYZE SDESC to ensure that the input descriptor
1645 1733 2 is valid. If it is, then ABUF will contain the address of the
1646 1734 2 first byte of the string, and A_LEN will contain its length.
1647 1735 2 -
1648 1736 2
1649 1737 2 STATUS = LIB$ANALYZE SDESC (.ADIGITS,A_LEN,ABUF);
1650 1738 2 IF .STATUS NEQ $$$_NORMAL
1651 1739 2 THEN
1652 1740 2 LIB$STOP (LIB$_INVARG);
1653 1741 2
1654 1742 2 +
1655 1743 2 Also check the BDIGITS descriptor before getting too involved
1656 1744 2 in this routine.
1657 1745 2 -
1658 1746 2
1659 1747 2 STATUS = LIB$ANALYZE SDESC (.BDIGITS,B_LEN,BBUF);
1660 1748 2 IF .STATUS NEQ $$$_NORMAL
1661 1749 2 THEN
1662 1750 2 LIB$STOP (LIB$_INVARG);
1663 1751 2
1664 1752 2 R_LEN = 0;
1665 1753 2 BEGIN
1666 1754 2
1667 1755 2 LOCAL
1668 1756 2 SCAN_DONE;
1669 1757 2
1670 1758 2 SCAN_DONE = 0;
1671 1759 2
1672 1760 2 DO
1673 1761 2 BEGIN
1674 1762 2
1675 1763 2 IF (.R_LEN EQLU .ADIGITS [DSC$W_LENGTH])
1676 1764 2 THEN
1677 1765 2 SCAN_DONE = 1
1678 1766 2 ELSE
1679 1767 2
1680 1768 2 IF ((.ABUF [.R_LEN] GEQ %'0') AND (.ABUF [.R_LEN] LEQ %'9'))
1681 1769 2 THEN
1682 1770 2 R_LEN = .R_LEN + 1
1683 1771 2 ELSE
1684 1772 2 SCAN_DONE = 1;
1685 1773 2
1686 1774 2 END
1687 1775 2 UNTIL (.SCAN_DONE);
1688 1776 2
1689 1777 2 END;
1690 1778 2 STR$GET1 DX (R_LEN, R_DESC);
1691 1779 2 RBUF = .R_DESC [DSC$A_POINTER];
1692 1780 2 CH$MOVE (.R_LEN, .ADIGITS [DSC$A_POINTER], RBUF [0]);
1693 1781 2
1694 1782 2 + Round or truncate the number if it has more than the desired number
```

```
1695 1783 2 | of significant digits.
1696 1784 2 |
1697 1785 2 |   RESULT_DIGITS = .R_LEN;
1698 1786 2 |
1699 1787 2 |   IF (.RESULT_DIGITS GTR ..PLACES)
1700 1788 2 |   THEN
1701 1789 2 |   BEGIN
1702 1790 3 |
1703 1791 4 |       IF ( NOT ..TRUNC)
1704 1792 3 |       THEN
1705 1793 4 |       BEGIN
1706 1794 4 |
1707 1795 4 | + Check the highest-order digit we will discard. If it is five or
1708 1796 4 | larger, round up. Note that the number is in sign-magnitude form
1709 1797 4 | at this point, so rounding "up" is always away from zero.
1710 1798 4 |
1711 1799 4 |
1712 1800 5 |       IF (.RBUF [..PLACES] GEQ %C'5')
1713 1801 4 |       THEN
1714 1802 5 |       BEGIN
1715 1803 5 |
1716 1804 5 |       LOCAL
1717 1805 5 |       CARRY_COUNTER,
1718 1806 5 |       CARRY_DONE;
1719 1807 5 |
1720 1808 5 |       CARRY_DONE = 0;
1721 1809 5 |       CARRY_COUNTER = ..PLACES - 1;
1722 1810 5 |
1723 1811 6 |       IF (.CARRY_COUNTER GEQ 0)
1724 1812 5 |       THEN
1725 1813 5 |
1726 1814 5 |       DO
1727 1815 6 |       BEGIN
1728 1816 6 |       RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] + 1;
1729 1817 6 |
1730 1818 7 |       IF (.RBUF [.CARRY_COUNTER] LEQ %C'9')
1731 1819 6 |       THEN
1732 1820 6 |       CARRY_DONE = 1
1733 1821 6 |       ELSE
1734 1822 7 |       BEGIN
1735 1823 7 |       RBUF [.CARRY_COUNTER] = .RBUF [.CARRY_COUNTER] - 10;
1736 1824 7 |       CARRY_COUNTER = .CARRY_COUNTER - 1;
1737 1825 6 |       END;
1738 1826 6 |
1739 1827 6 |       END
1740 1828 5 |       UNTIL ((.CARRY_DONE) OR (.CARRY_COUNTER LSS 0));
1741 1829 5 |
1742 1830 6 |       IF ( NOT .CARRY_DONE)
1743 1831 5 |       THEN
1744 1832 6 |       BEGIN
1745 1833 6 |
1746 1834 6 | + The carry has forced a right shift (all 9's rounded up).
1747 1835 6 | We are guaranteed enough space since we must be dropping at least
1748 1836 6 | one digit. Because of this shift we must now be dropping at least
1749 1837 6 | two digits.
1750 1838 6 |
1751 1839 6 |
```

```
1752 1840 6      INCR COUNTER FROM 0 TO ..PLACES - 2 DO
1753 1841 6      RBUF [.COUNTER + 1] = .RBUF [.COUNTER];
1754 1842 6
1755 1843 6      RBUF [0] = %C'1';
1756 1844 6      REXP = .REXP + 1;
1757 1845 6      END;
1758 1846 5
1759 1847 4      END;
1760 1848 4
1761 1849 3      END;
1762 1850 3
1763 1851 2      REXP = .REXP + (.RESULT DIGITS - ..PLACES);
1764 1852 2      RESULT_DIGITS = ..PLACES;
1765 1853 2      END;
1766 1854 2
1767 1855 2
1768 1856 2      + Return the results to the caller in the B operand.
1769 1857 2      - If there are no digits left, return a single zero digit.
1770 1858 2
1771 1859 2
1772 1860 2      IF (.RESULT_DIGITS EQL 0)
1773 1861 2      THEN
1774 1862 2          BEGIN
1775 1863 2              .BSIGN = 0;
1776 1864 2              .BEXP = 0;
1777 1865 2              STR$COPY_R (.BDIGITS, %REF (1), %REF (%ASCII'0'));
1778 1866 2              CHK_STR_TYPE (.BDIGITS[DSC$A_POINTER],%REF (1),.BDIGITS);
1779 1867 2          END
1780 1868 2
1781 1869 2      ELSE
1782 1870 2
1783 1871 2      + Call CHK_STR_TYPE to determine if we need to pad the number with
1784 1872 2      - leading zeroes depending on the string type.
1785 1873 2
1786 1874 2
1787 1875 2
1788 1876 2      BEGIN
1789 1877 2          .BSIGN = .A SIGN;
1790 1878 2          .BEXP = .REXP;
1791 1879 2          CHK_STR_TYPE (.R_DESC[DSC$A_POINTER],RESULT_DIGITS,.BDIGITS);
1792 1880 2          END;
1793 1881 2
1794 1882 2      BEGIN
1795 1883 2          .BSIGN = .A SIGN;
1796 1884 2          .BEXP = .REXP;
1797 1885 2          STR$COPY_R (.BDIGITS, RESULT_DIGITS, .R_DESC [DSC$A_POINTER]);
1798 1886 2          END;
1799 1887 2
1800 1888 2      + Free our string.
1801 1889 2      -
1802 1890 2      STR$FREE1_DX (R_DESC);
1803 1891 2      RETURN;
1804 1892 2      END;
1805 1893 2
```

! end of STR\$ROUND

00 00 00 44 4E 55 4F 52 24 52 54 53 00B52
00B54 P.AAI: .BLKB 2
.ASCII \STR\$ROUND\<0><0><0>

				OFFC	00000	.ENTRY	STR\$ROUND, Save R2,R3,R4,R5,R6,R7,R8,R9,-	
	5B	00000000G	00	9E	00002	MOVAB	R10,R11	1612
	5A	00000000G	00	9E	00009	MOVAB	LIB\$ANALYZE_SDESC, R11	
	5E		30	C2	00010	SUBL2	LIB\$STOP, R10	
		28	AE	7C	00013	CLRQ	#48, SP	
	6D	016C	CF	DE	00016	MOVAL	R_DESC	1664
	0B		6C	91	0001B	CMQB	17\$, (FP)	
			1E	1E	0001E	BGEQU	(AP), #8	1707
20	AE	010E0009	8F	D0	00020	MOVL	1\$	
24	AE	C9	AF	9E	00028	MOVL	#17694729, ROUT_NAME_DESC	1714
		20	AE	9F	0002D	MOVAB	P.AAI, ROUT_NAME_DESC+4	1717
	7E		6C	9A	00030	PUSHAB	ROUT_NAME_DESC	1718
		00000000G	02	DD	00033	MOVZBL	(AP), -(SP)	
	6A		8F	DD	00035	PUSHL	#2	
	59	0C	04	FB	0003B	PUSHL	#STR\$ WRONUMARG	
	58	10	BC	D0	0003E	CALLS	#4, LIB\$STOP	
		28	AE	B4	00046	MOVL	@ASIGN, A SIGN	1724
2A	AE		0F	90	00049	MOVL	@AEXP, REXP	1725
2B	AE		02	90	0004D	CLRQ	R_DESC	1726
		2C	AE	D4	00051	MOVB	#TS, R_DESC+2	1727
		08	AE	9F	00054	MOVB	#2, R_DESC+3	1728
		10	AE	9F	00057	CLRL	R_DESC+4	1729
	52	14	AC	D0	0005A	PUSHAB	ABUF	1737
			52	DD	0005E	PUSHAB	A_LEN	
	6B		03	FB	00060	MOVL	ADIGITS, R2	
	53		50	D0	00063	PUSHL	R2	
	01		53	D1	00066	CALLS	#3, LIB\$ANALYZE_SDESC	
		00000000G	09	13	00069	MOVL	R0, STATUS	
	6A		8F	DD	0006B	CMPL	STATUS, #1	1738
		10	01	FB	00071	BEQL	2\$	
		18	AE	9F	00074	PUSHL	#LIB\$ INVARG	1740
		20	AE	9F	00077	CALLS	#1, LIB\$STOP	
	57		AC	D0	0007A	PUSHAB	BBUF	1747
			57	DD	0007E	PUSHAB	B_LEN	
	6B		03	FB	00080	MOVL	BDIGITS, R7	
	53		50	D0	00083	PUSHL	R7	
	01		53	D1	00086	CALLS	#3, LIB\$ANALYZE_SDESC	
		00000000G	09	13	00089	MOVL	R0, STATUS	
	6A		8F	DD	0008B	CMPL	STATUS, #1	1748
		18	01	FB	00091	BEQL	3\$	
			AE	D4	00094	PUSHL	#LIB\$ INVARG	1750
			51	D4	00097	CALLS	#1, LIB\$STOP	
18	AE	62	00	ED	00099	CLRL	R_LEN	1752
		50	15	13	0009F	CLRL	SCAN DONE	1758
		08	AE	C1	000A1	CMPZV	#0, #16, (R2), R_LEN	1763
	30		60	91	000A7	BEQL	5\$	
			0A	1F	000AA	ADDL3	R_LEN, ABUF, R0	1768
	39		60	91	000AC	CMQB	(R0), #48	
			05	1A	000AF	BLSSU	5\$	
			18	AE	000B1	CMQB	(R0), #57	
						BGTRU	5\$	
						INCL	R_LEN	1770

			03	11	000B4	BRB	6\$			
	51		01	D0	000B6	5\$:	MOVL	#1, SCAN DONE	1772	
	DD		51	E9	000B9	6\$:	BLBC	SCAN DONE, 4\$	1775	
		28	AE	9F	000BC		PUSHAB	R_DESC	1778	
		1C	AE	9F	000BF		PUSHAB	R_LEN		
00000000G	00		02	FB	000C2		CALLS	#2, STR\$GET1_DX		
	56		AE	D0	000C9		MOVL	R_DESC+4, RBUF	1779	
66	04	B2	18	AE	28	000CD	MOVCL	R_LEN, @4(R2), (RBUF)	1780	
	1C	AE	18	AE	D0	000D3	MOVL	R_LEN, RESULT_DIGITS	1785	
		51	04	BC	D0	000D8	MOVL	@PLACES, R1	1787	
		51	1C	AE	D1	000DC	CMPL	RESULT_DIGITS, R1		
			54	15	000E0		BLEQ	14\$		
	44		08	BC	E8	000E2	BLBS	@TRUNC, 13\$	1791	
	35		6146	91	000E6		CMPB	(R1)[RBUF], #53	1800	
			3E	1F	000EA		BLSSU	13\$		
			52	D4	000EC		CLRL	CARRY_DONE	1808	
	50		FF	A1	9E	000EE	MOVAB	-1(R1), CARRY_COUNTER	1809	
			1B	19	000F2		BLSS	10\$	1811	
			6046	96	000F4	7\$:	INCB	(CARRY_COUNTER)[RBUF]	1816	
	39		6046	91	000F7		CMPB	(CARRY_COUNTER)[RBUF], #57	1818	
			05	1A	000FB		BGTRU	8\$		
	52		01	D0	000FD		MOVL	#1, CARRY_DONE	1820	
			06	11	00100		BRB	9\$		
	6046		0A	82	00102	8\$:	SUBB2	#10, (CARRY_COUNTER)[RBUF]	1823	
			50	D7	00106		DECL	CARRY_COUNTER	1824	
	1F		52	E8	00108	9\$:	BLBS	CARRY_DONE, 13\$	1828	
			50	D5	0010B		TSTL	CARRY_COUNTER		
			E5	18	0010D		BGEQ	7\$		
	18		52	E8	0010F	10\$:	BLBS	CARRY_DONE, 13\$	1830	
	50		FE	A1	9E	00112	MOVAB	-2(R1), R0	1840	
	52		01	CE	00116		MNEGL	#1, COUNTER		
			06	11	00119		BRB	12\$		
	01	A246	6246	90	0011B	11\$:	MOVB	(COUNTER)[RBUF], 1(COUNTER)[RBUF]	1841	
F6		52	50	F3	00121	12\$:	AOBLEQ	R0, COUNTER, 11\$		
		66	31	90	00125		MOVB	#49, (RBUF)	1843	
			58	D6	00128		INCL	REXP	1844	
50	1C	AE	51	C3	0012A	13\$:	SUBL3	R1, RESULT_DIGITS, R0	1851	
		58	50	C0	0012F		ADDL2	R0, REXP		
	1C	AE	51	D0	00132		MOVL	R1, RESULT_DIGITS	1852	
			1C	AE	D5	00136	14\$:	TSTL	RESULT_DIGITS	1860
			2B	12	00139		BNEQ	15\$		
			18	BC	D4	0013B	CLRL	@BSIGN	1863	
			1C	BC	D4	0013E	CLRL	@BEXP	1864	
	04	AE	30	D0	00141		MOVL	#48, 4(SP)	1865	
			04	AE	9F	00145	PUSHAB	4(SP)		
	04	AE	01	D0	00148		MOVL	#1, 4(SP)		
			04	AE	9F	0014C	PUSHAB	4(SP)		
			57	DD	0014F		PUSHL	R7		
00000000G	00		03	FB	00151		CALLS	#3, STR\$COPY_R		
			57	DD	00158		PUSHL	R7	1866	
	08	AE	01	D0	0015A		MOVL	#1, 8(SP)		
			08	AE	9F	0015E	PUSHAB	8(SP)		
			04	A7	DD	00161	PUSHL	4(R7)		
			10	11	00164		BRB	16\$		
	18	BC	59	D0	00166	15\$:	MOVL	A SIGN, @BSIGN	1877	
	1C	BC	58	D0	0016A		MOVL	REXP, @BEXP	1878	
			57	DD	0016E		PUSHL	R7	1879	

STR\$ARITH
1-019

H 14
16-Sep-1984 01:27:51
14-Sep-1984 12:40:01

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]STRARITH.B32;1

Page 52
(7)

		20	AE	9F	00170		PUSHAB	RESULT DIGITS
		34	AE	DD	00173		PUSHL	R_DESC74
0000V	CF		03	FB	00176	16\$:	CALLS	#3, CHK_STR_TYPE
		28	AE	9F	00178		PUSHAB	R_DESC
00000000G	00		01	FB	0017E		CALLS	#T, STR\$FREE1_DX
				04	00185		RET	
				0000	00186	17\$:	.WORD	Save nothing
	50	08	AC	D0	00188		MOVL	8(AP), R0
	50	04	A0	D0	0018C		MOVL	4(R0), R0
		FB	A0	9F	00190		PUSHAB	R_DESC
			01	DD	00193		PUSHL	#T
			5E	DD	00195		PUSHL	SP
	7E	04	AC	7D	00197		MOVQ	4(AP), -(SP)
0000V	CF		03	FB	00198		CALLS	#3, FREE_STRINGS
				04	001A0		RET	

1891
1893
1664

; Routine Size: 417 bytes, Routine Base: _STR\$CODE + 0B60

; 1806 1894 1

```
1808 1895 1 GLOBAL ROUTINE STR$DIVIDE (
1809 1896 1     ASIGN,
1810 1897 1     AEXP,
1811 1898 1     ADIGITS,
1812 1899 1     BSIGN,
1813 1900 1     BEXP,
1814 1901 1     BDIGITS,
1815 1902 1     TOT_DIGITS,
1816 1903 1
1817 1904 1     RND_TRUNC,
1818 1905 1     CSIGN,
1819 1906 1     CEXP,
1820 1907 1     CDIGITS ):NOVALUE =
```

```
! Sign of operand A
! Decimal exponent of operand A
! Digits of operand A
! Sign of operand B
! Decimal exponent of operand B
! Digits of operand B
! Number of digits to the right of the
! decimal point to carry out the divide
! Round/Truncate parameter
! To contain sign of operand C
! To contain decimal exponent of operand C
! To contain digits of operand C
```


1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878

1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964

++

FUNCTIONAL DESCRIPTION:

This routine finds the quotient of two decimal strings i.e. $C = A / B$.
The algorithm implemented here has been provided by KNUTH. It is
his famous Algorithm D (division of non-negative integers (which has
been modified to handle negative integers)) found in Volume 2 of
that extraordinary series (Vol. 2 is entitled Seminumerical Algorithms).
An explanation of the algorithm appears further on in the program.

CALLING SEQUENCE:

STR\$DIVIDE (ASIGN.rv.r,AEXP.rl.r,ADIGITS.rnu.dx,
BSIGN.rv.r,BEXP.rl.r,BDIGITS.rnu.dx,
TOT_DIGITS.rl.r,RND_TRUNC.rv.r,
CSIGN.wv.r,CEXP.wl.r,CDIGITS.wnu.dx)

FORMAL PARAMETERS:

ASIGN.rv.r	Sign of operand A (0=positive, 1=negative)
AEXP.rl.r	Power of 10 by which to multiply the operand A digits to get the absolute value of operand A. Ex: AEXP=1,ADIGITS=123 gives 1230.
ADIGITS.rnu.dx	Descriptor for the digits of operand A.
BSIGN.rv.r	Sign of operand B (0=positive, 1=negative)
BEXP.rl.r	Power of 10 by which to multiply the operand B digits to get the absolute value of operand B. Ex: BEXP=-1,BDIGITS=123 gives 12.3
BDIGITS.rnu.dx	Descriptor for the digits of operand B.
TOT_DIGITS.rl.r	Number of digits to the right of the decimal point to carry out the division
RND_TRUNC.rv.r	Round/Truncate parameter (0 = truncate, 1 = round)
CSIGN.wv.r	Sign of operand C (0=positive, 1=negative)
CEXP.wl.r	Power of 10 by which to multiply the operand C digits to get the absolute value of operand C. Ex: CEXP=0,CDIGITS=123 gives 123.
CDIGITS.wnu.dx	Descriptor for the digits of operand C.

IMPLICIT INPUTS:

-NONE

IMPLICIT OUTPUTS:

-CSIGN
-CEXP
-CDIGITS

ROUTINE VALUE:

-NONE

COMPLETION CODES

-NONE

STRARITH
1-019

K 14
16-Sep-1984 01:27:51
14-Sep-1984 12:40:01

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]STRARITH.B32;1

Page 55
(9)

1879	1965	1	MACROS:
1880	1966	1	
1881	1967	1	-NONE
1882	1968	1	
1883	1969	1	SIDE EFFECTS:
1884	1970	1	
1885	1971	1	Signals if storage is exceeded.
1886	1972	1	
1887	1973	1	-

```
1889 1974 2 BEGIN
1890 1975
1891 1976
1892 1977 MAP
1893 1978 ADIGITS:REF BLOCK [8,BYTE],
1894 1979 BDIGITS:REF BLOCK [8,BYTE],
1895 1980 CDIGITS:REF BLOCK [8,BYTE];
1896 1981
1897 1982 STACKLOCAL
1898 1983 A_LENGTH:WORD,      Number of digits in A string
1899 1984 A_ADDR,          Address of A string
1900 1985 B_LENGTH:WORD,      Number of digits in B string
1901 1986 B_ADDR,          Address of B string
1902 1987 C_LENGTH:WORD,      Number of digits in result string
1903 1988 START_BUF,        Pointer to 1st byte of allocated memory
1904 1989 A_LEN,           Number of digits needed in A to compute result
1905 1990 A_CHUNKS,         Number of 15 digit chunks needed in A
1906 1991 ABUF,            Pointer to packed decimal chunks of A
1907 1992 B_CHUNKS,         Number of 15 digit chunks in B
1908 1993 BBUF,            Pointer to packed decimal chunks of B
1909 1994 DRBUF,           Pointer to D / pointer to rounding chunk
1910 1995 Q_LENGTH,        Number of digit required in the quotient
1911 1996 Q_CHUNKS,         Number of 15 digit chunks required in quotient
1912 1997 QBUF,            Pointer to 15 digit chunks of the quotient
1913 1998 QSTRBUF,          Pointer to string of quotient digits
1914 1999 QBBUF,           Pointer to buffer containing q*B
1915 2000 STATUS,          Longword for returning status
1916 2001 FLAG,            B_CHUNKS = 1 ==> FLAG = 1, FLAG = 0 otherwise
1917 2002 TEMP,            Longword to hold intermediate results
1918 2003 LEADING_ZEROS,    Number of leading zeros in the quotient string
1919 2004 STORAGE: VECTOR[4,BYTE] INITIAL(%C'0'),
1920 2005 DUMMY_STORAGE,    Dummy storage area
1921 2006 BYTES_VM;         Total bytes of storage allocated
```

```
1923 2007 2 *****
1924 2008 2
1925 2009 2
1926 2010 2
1927 2011 2
1928 2012 2
1929 2013 2
1930 2014 2
1931 2015 2
1932 2016 2
1933 2017 2
1934 2018 2
1935 2019 2
1936 2020 2
1937 2021 2
1938 2022 2
1939 2023 2
1940 2024 2
1941 2025 2
1942 2026 2
1943 2027 2
1944 2028 2
1945 2029 2
1946 2030 2
1947 2031 2
1948 2032 2
1949 2033 2
1950 2034 2
1951 2035 2
1952 2036 2
1953 2037 2
1954 2038 2
1955 2039 2
1956 2040 2
1957 2041 2
1958 2042 2
1959 2043 2
1960 2044 2
1961 2045 2
1962 2046 2
1963 2047 2
1964 2048 2
1965 2049 2
1966 2050 2
1967 2051 2
1968 2052 2
1969 2053 2
1970 2054 2
1971 2055 2
```

THE ALGORITHM

GIVENS: n = length of the divisor
 m = length of dividend - n
radix = 10 (decimal)

STEP 1. Normalize. Set $D = \text{FLOOR}(\text{radix}/(v_1+1))$ where v_1 is the first digit of the divisor which must not be zero. Where $U_0 U_1 \dots U_{m+n}$ represent the chunks of 15 digits of the dividend and $V_1 V_2 \dots V_n$ represent the chunks of 15 digits of the divisor. Multiply A by D thus giving the sequence of 15 digit chunks $U_0 U_1 U_2 \dots U_{m+n}$. (Note the introduction of the new chunk.) Multiply B by d to obtain a sequence of chunks $V_1 V_2 \dots V_n$. (Note no new chunk occurs)

STEP 2. Set $J = 0$. This is the value we will loop on. For this routine we will loop "LOOP" number of times. Steps 2-7 will provide the basis for the division of $U_j U_{j+1} \dots U_{j+n}$ by $V_1 V_2 \dots V_n$, to get a single quotient digit - Q_j .

STEP 3. Calculate the first digit of the quotient:
If $U_j = V_1$ then set $q = \text{radix} - 1$. Otherwise, set $q = \text{FLOOR}((U_j * \text{radix} + U_{j+1}) / V_1)$. Now test if $V_2 * q > ((U_j * \text{radix} + U_{j+1} - q * V_1) * \text{radix}) + U_{j+2}$. If so, then decrease q by 1 and repeat this test. When finish q is either equal to the quotient digit or one greater.

STEP 4. Multiply and subtract. Replace $U_j U_{j+1} \dots U_{j+n}$ by $U_j U_{j+1} \dots U_{j+n} - (q * V_1 V_2 \dots V_n)$. This step consists of a simple multiplication by a one-place number, combined with a subtraction. The digits $U_j U_{j+1} \dots U_{j+n}$ should be kept positive; if the result of this step is negative, $U_j U_{j+1} \dots U_{j+n}$ should be left as the true value plus radix raised to the $n+1$, i.e. as the radix complement of the true value, and a "borrow" to the left should be remembered.

STEP 5. Set $Q[J] = q$. This is a digit of the quotient. If the result of STEP 4 was negative, go to STEP 6; otherwise go to STEP 7.

STEP 6. Decrease $Q[J]$ by 1. Add $0V_1 V_2 \dots V_n$ to $U_j U_{j+1} \dots U_{j+n}$.

STEP 7. Loop on J . If $J \leq \text{"LOOP"}$ then go back to STEP 3.


```
1973 2056 2 1+
1974 2057 2 1- Validate input descriptors.
1975 2058 2
1976 2059 2 STATUS = LIB$ANALYZE_SDESC (.ADIGITS, A_LENGTH, A_ADDR);
1977 2060 2 IF .STATUS NEQ SSS_NORMAL
1978 2061 2 THEN
1979 2062 2 LIB$STOP (LIB$INVARG); ! Unsuccessful status
1980 2063 2
1981 2064 2 STATUS = LIB$ANALYZE_SDESC (.BDIGITS, B_LENGTH, B_ADDR);
1982 2065 2 IF .STATUS NEQ SSS_NORMAL
1983 2066 2 THEN
1984 2067 2 LIB$STOP (LIB$INVARG); ! Unsuccessful status
1985 2068 2
1986 2069 2 STATUS = LIB$ANALYZE_SDESC (.CDIGITS, C_LENGTH, TEMP);
1987 2070 2 IF .STATUS NEQ SSS_NORMAL
1988 2071 2 THEN
1989 2072 2 LIB$STOP (LIB$INVARG);
1990 2073 2 1+
1991 2074 2 1- Validate input strings - If SPANC returns a zero value all characters are
1992 2075 2 digits. If it returns a non-zero value, then TEMP is the address of the
1993 2076 2 first non-digit.
1994 2077 2
1995 2078 2 TEMP = SPANC (A_LENGTH, .A_ADDR, SPANC_TABLE, MASK);
1996 2079 2 IF .TEMP NEQ 0 ! No match found
1997 2080 2 THEN
1998 2081 2 A_LENGTH = .TEMP - .A_ADDR;
1999 2082 2
2000 2083 2 TEMP = SPANC (B_LENGTH, .B_ADDR, SPANC_TABLE, MASK);
2001 2084 2 IF .TEMP NEQ 0 ! No match found
2002 2085 2 THEN
2003 2086 2 B_LENGTH = .TEMP - .B_ADDR;
```

```
2005 2087 +
2006 2088 *** BEGIN THE DIVISION ALGORITHM ***
2007 2089
2008 2090
2009 2091 +
2010 2092 Calculate the resultant sign and exponent.
2011 2093 -
2012 2094 .CSIGN = ..ASIGN XOR ..BSIGN;
2013 2095 .CEXP = -..TOT_DIGITS;
2014 2096
2015 2097 +
2016 2098 Strip off leading zeros for A and B and compute their length.
2017 2099 CH$FIND_NOT_CH returns a null pointer if the desired match on character
2018 2100 is not found. To determine if the pointer is null or not,
2019 2101 one must invoke CH$FAIL which returns a value of one if the pointer
2020 2102 is null, and a zero if it is not null.
2021 2103 -
2022 2104 TEMP = CH$FIND_NOT_CH (.A_LENGTH, .A_ADDR, %C'0');
2023 2105 STATUS = CH$FAIL (.TEMP);
2024 2106 IF .STATUS EQL 0
2025 2107 THEN
2026 2108 BEGIN
2027 2109 A_LENGTH = .A_LENGTH - (.TEMP - .A_ADDR);
2028 2110 A_ADDR = .TEMP;
2029 2111 END
2030 2112 ELSE
2031 2113 .CSIGN = 0;
2032 2114
2033 2115 TEMP = CH$FIND_NOT_CH (.B_LENGTH, .B_ADDR, %C'0');
2034 2116 STATUS = CH$FAIL (.TEMP);
2035 2117 IF .STATUS EQL 1
2036 2118 THEN
2037 2119 LIB$STOP (STR$ DIVBY_ZER);
2038 2120 B_LENGTH = .B_LENGTH - (.TEMP - .B_ADDR);
2039 2121 B_ADDR = .TEMP;
2040 2122
2041 2123 +
2042 2124 Calculate maximum number of result digits required
2043 2125 +
2044 2126 Q_LENGTH = (.A_LENGTH + ..AEXP) - (.B_LENGTH + ..BEXP)
2045 2127 + ..TOT_DIGITS + ..RND_TRUNC;
2046 2128
2047 2129 IF .Q_LENGTH LSS 0
2048 2130 THEN
2049 2131 +
2050 2132 Special case for zero quotient
2051 2133 -
2052 2134 BEGIN
2053 2135 LEADING_ZEROS = 0;
2054 2136 BYTES_VM = MAXU(.Q_LENGTH, 1);
2055 2137 STATUS = LIB$GET_VM (BYTES_VM, START_BUF);
2056 2138 QSTRBUF = STORAGE;
2057 2139 END
2058 2140 ELSE
2059 2141 BEGIN
2060 2142 +
2061 2143 Determine the number of digits required in A to obtain the proper number
```

```
2062 2144 of digits in the result
2063 2145
2064 2146 A_LEN = .B_LENGTH + .Q_LENGTH;
2065 2147
2066 2148 + Determine the number of 15 digit CHUNKS needed to hold B, the required
2067 2149 digits of A and the result digits
2068 2150
2069 2151 A_CHUNKS = (.A_LEN + 14)/15;
2070 2152 B_CHUNKS = (.B_LENGTH + 14)/15;
2071 2153 Q_CHUNKS = (.Q_LENGTH + 29)/15;
2072 2154
2073 2155 + For the algorithm we must have A_CHUNKS >= B_CHUNKS + Q_CHUNKS.
2074 2156
2075 2157 A_CHUNKS = MAXU(.A_CHUNKS, .B_CHUNKS + .Q_CHUNKS);
2076 2158
2077 2159 + Determine total storage needed as the maximum of (the storage needed for
2078 2160 the computation of the quotient in packed decimal, the storage needed to
2079 2161 convert the quotient to a string, the length of the result string)
2080 2162
2081 2163 BYTES_VM = 8*(.B_CHUNKS*2 + .A_CHUNKS + 3); # of bytes to perform
2082 2164 division algorithm in
2083 2165 packed decimal.
2084 2166 TEMP = (.Q_CHUNKS + 1) * 15; # of bytes needed to hold
2085 2167 quotient string
2086 2168 BYTES_VM = MAXU (.BYTES_VM, .TEMP + .C_LENGTH);
2087 2169 Need .TEMP+.C_LENGTH
2088 2170 here to ensure string
2089 2171 is long enough for case of
2090 2172 zero padding of fixed
2091 2173 length strings.
2092 2174
2093 2175 + Allocate working storage and set up pointers into it.
2094 2176
2095 2177 STATUS = LIB$GET_VM (BYTES_VM, START_BUF);
2096 2178 DRBUF = .START_BUF;
2097 2179 QBUF = .DRBUF;
2098 2180 ABUF = .QBUF + 8;
2099 2181 BBUF = .ABUF + (.A_CHUNKS + 1)*8;
2100 2182 QBBUF = .BBUF + .B_CHUNKS*8;
2101 2183 QSTRBUF = .START_BUF + .BYTES_VM - .Q_CHUNKS*15;
2102 2184
2103 2185 + Convert A and B strings to packed decimal.
2104 2186
2105 2187 LIB$SCVT_STR_PACK R9 (.A_ADDR, .A_LENGTH, .A_CHUNKS, .ABUF + 8);
2106 2188 MOVP (%REF(15), ZERO, .ABUF);
2107 2189 LIB$SCVT_STR_PACK R9 (.B_ADDR, .B_LENGTH, .B_CHUNKS, .BBUF);
```

```
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131

2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212

+ Step 1 - Normalize A and B. NOTE: If B_CHUNKS = 1 this step is not necessary
- and the computation of q can be simplified. A flag is used to indicate the
  proper method of evaluating q. FLAG = 1 if .B_CHUNKS = 1 and 0 otherwise.

      IF .B_CHUNKS NEQ 1
      THEN
      BEGIN
      FLAG = 0;
      STATUS = LIB$SCALC_D_R7 (.BBUF, .DRBUF);
      IF .STATUS NEQ 1 ! STATUS = 1 <==> D = 1
      THEN
      BEGIN
      LIB$MUL_PACK_R10 (.DRBUF, .ABUF + 8, .A_CHUNKS,
      .A_CHUNKS + 1, .ABUF + 8);
      LIB$MUL_PACK_R10 (.DRBUF, .BBUF, .B_CHUNKS,
      .B_CHUNKS, .BBUF);
      END
      ELSE
      MOVP (XREF(15), ZERO, .ABUF);
      END
      ELSE
      FLAG = 1;
```



```
2133 2213 3 + Ready to start the actual divide algorithm.
2134 2214 4
2135 2215 4 - INCR J FROM 0 TO (.Q_CHUNKS*8 - 8) BY 8 DO
2136 2216 4 BEGIN
2137 2217 4
2138 2218 4 + Step 3 - Calculate digit of quotient.
2139 2219 4 -
2140 2220 4 STATUS = LIB$CALC_Q_R9 (.BBUF, .ABUF + .J, .FLAG, .QBUF + .J);
2141 2221 4 IF .STATUS NEQ 1
2142 2222 4 THEN
2143 2223 4 LIB$STOP (LIB$INVARG);
2144 2224 4
2145 2225 4 + Step 4 - Multiply and subtract. Replace the digits of ABUF by ABUF - Q*BBUF
2146 2226 4 -
2147 2227 4 LIB$MUL_PACK_R10 (.QBUF + .J, .BBUF, .B_CHUNKS,
2148 2228 4 .B_CHUNKS+1, .QBBUF + 8);
2149 2229 4 STATUS = LIB$SUB_PACK_R8 (.B_CHUNKS, .ABUF + .J, .QBBUF);
2150 2230 4
2151 2231 4 + Step 6 - Adjust q if the result of step 4 was negative
2152 2232 4 -
2153 2233 4 IF .STATUS EQL 1 ! If remainder is negative
2154 2234 4 THEN
2155 2235 4 LIB$ADJUST_Q_R9 (.B_CHUNKS, .ABUF + .J + 8, .BBUF, .QBUF + .J);
2156 2236 4
2157 2237 4 END;
2158 2238 4
2159 2239 4 + Check if rounding is required and round result if necessary
2160 2240 4 -
2161 2241 4 IF ..RND_TRUNC EQL 1
2162 2242 4 THEN
2163 2243 4 BEGIN
2164 2244 4 TEMP = (.Q_CHUNKS-1)*15 - .Q_LENGTH;
2165 2245 4 Q_LENGTH = .Q_LENGTH - 1;
2166 2246 4 DRBUF = .QBUF + (.Q_CHUNKS - 1)*8;
2167 2247 4 LIB$ROUND_R7 (.DRBUF, .TEMP);
2168 2248 4 END;
2169 2249 4
2170 2250 4 + Check if 1st chunk of the quotient is zero. If it is, A < B, the number of
2171 2251 4 leading zeros is 15. Otherwise, see if its less than 10. if it is, then
2172 2252 4 the number of leading zeros is 14 and the number of digits in the quotient
2173 2253 4 should be increased by 1. Otherwise, the number of leading zeros is 13 and
2174 2254 4 the number of digits in the quotient should be increased by 2.
2175 2255 4 -
2176 2256 4 STATUS = CMPP (XREF(15), .QBUF, XREF(15), ZERO);
2177 2257 4 IF .STATUS EQL 0
2178 2258 4 THEN
2179 2259 4 LEADING_ZEROS = 15
2180 2260 4 ELSE
2181 2261 4 BEGIN
2182 2262 4 STATUS = CMPP (XREF(15), .QBUF, XREF(15), TEN);
2183 2263 4 IF .STATUS LSS 0
2184 2264 4 THEN
2185 2265 4 BEGIN
2186 2266 4 Q_LENGTH = .Q_LENGTH + 1;
2187 2267 4 LEADING_ZEROS = 14;
2188 2268 4 END
2189 2269 4 ELSE
```

```
2190      BEGIN
2191      Q_LENGTH = .Q_LENGTH + 2;
2192      LEADING_ZEROS = 13;
2193      END;
2194      END;
2195
2196      + Convert the packed number back into its original numeric form.
2197      - LIB$SCVT_PACK_STR_R8 (.QBUF, .Q_CHUNKS, .QSTRBUF);
2198      END;
2199
2200      + Check descriptor class to see if the string needs to be padded with leading
2201      - zeros before copying the quotient string to the result string.
2202      IF (.CDIGITS[DSC$B_CLASS] NEQ DSC$K_CLASS_D) AND
2203      (.CDIGITS[DSC$B_CLASS] NEQ DSC$K_CLASS_VS) AND
2204      (.C_LENGTH GTR .Q_LENGTH)
2205      THEN
2206      BEGIN
2207      TEMP = .C_LENGTH - .Q_LENGTH - .LEADING_ZEROS;
2208      Q_LENGTH = .C_LENGTH;
2209      QSTRBUF = .QSTRBUF - .TEMP;
2210      IF .TEMP GEQ 0
2211      THEN
2212      CH$FILL ('0', .TEMP, .QSTRBUF);
2213      END
2214      ELSE
2215      QSTRBUF = .QSTRBUF + .LEADING_ZEROS;
2216
2217      + Check the type of descriptor our resultant descriptor is.
2218      - QSTRBUF = .QSTRBUF + .LEADING_ZEROS;
2219      - CHK_STR_TYPE (QSTRBUF, Q_LENGTH, .CDIGITS);
2220
2221      + Copy quotient string to result string and deallocate virtual memory.
2222      - IF .Q_LENGTH LEQ 0
2223      THEN
2224      STATUS = LIB$SCOPY_R_DX (%REF(1), %REF (%ASCII'0'), .CDIGITS)
2225      ELSE
2226      STATUS = LIB$SCOPY_R_DX (Q_LENGTH, .QSTRBUF, .CDIGITS);
2227      STATUS = LIB$FREE_VM (BYTES_VM, START_BUF);
2228      END;
```

OFFC 00000
SE 94 AE 9E 00002

.ENTRY STR\$DIVIDE, Save R2,R3,R4,R5,R6,R7,R8,R9,-
R10,R11
MOVAB -108(SP), SP

: 1895
:
:

			14	AE		30	DO	00006	MOVL	#48, STORAGE	1974
			64	AE	6E	AE	9F	0000A	PUSHAB	A_ADDR	2059
			6E	AE	0C	AC	DD	00010	PUSHAB	A_LENGTH	
			00000000G	00		03	FB	00013	PUSHL	ADIGITS	
			24	AE		50	DO	0001A	CALLS	#3, LIB\$ANALYZE_SDESC	
			01	01	24	AE	D1	0001E	MOVL	R0, STATUS	2060
						0D	13	00022	CMPL	STATUS, #1	
			00000000G	00	00000000G	8F	DD	00024	BEQL	1\$	2062
					5C	01	FB	0002A	PUSHL	#LIB\$ INVARG	
					66	AE	9F	00031	CALLS	#1, LIB\$STOP	2064
					18	AE	9F	00034	PUSHAB	B_ADDR	
			00000000G	00		AC	DD	00037	PUSHAB	B_LENGTH	
			24	AE		03	FB	0003A	PUSHL	BDIGITS	
			01	01	24	50	DO	00041	CALLS	#3, LIB\$ANALYZE_SDESC	
						AE	D1	00045	MOVL	R0, STATUS	2065
						0D	13	00049	CMPL	STATUS, #1	
			00000000G	DO	00000000G	8F	DD	0004B	BEQL	2\$	2067
					1C	01	FB	00051	PUSHL	#LIB\$ INVARG	
					5E	AE	9F	00058	CALLS	#1, LIB\$STOP	2069
					2C	AE	9F	0005B	PUSHAB	TEMP	
			00000000G	00		AC	DD	0005E	PUSHAB	C_LENGTH	
			24	AE		03	FB	00061	PUSHL	CDIGITS	
			01	01	24	50	DO	00068	CALLS	#3, LIB\$ANALYZE_SDESC	
						AE	D1	0006C	MOVL	R0, STATUS	2070
						0D	13	00070	CMPL	STATUS, #1	
			00000000G	00	00000000G	8F	DD	00072	BEQL	3\$	2072
			64	BE	6A	01	FB	00078	PUSHL	#LIB\$ INVARG	
F385	CF	F288	CF			AE	2B	0007F	CALLS	#1, LIB\$STOP	2078
						02	12	0008A	SPANC	A_LENGTH, @A_ADDR, SPANC_TABLE, MASK	
						51	D4	0008C	BNEQ	4\$	
			1C	AE		51	DO	0008E	CLRL	R1	
						07	13	00092	MOVL	R1, TEMP	2079
			6A	AE	64	AE	A3	00094	BEQL	5\$	2081
F369	CF	F26C	CF	1C	AE	AE	A3	00094	SUBW3	A_ADDR, TEMP, A_LENGTH	
				5C	BE	62	AE	2B	SPANC	B_LENGTH, @B_ADDR, SPANC_TABLE, MASK	2083
						02	12	000A6	BNEQ	6\$	
						51	D4	000A8	CLRL	R1	
			1C	AE		51	DO	000AA	MOVL	R1, TEMP	
						07	13	000AE	BEQL	7\$	2084
			62	AE	5C	AE	A3	000B0	SUBW3	B_ADDR, TEMP, B_LENGTH	2086
			24	BC	10	BC	CD	000B7	XORL3	@BSIGN, @ASIGN, @CSIGN	2094
					1C	BC	DO	000BE	MOVL	@TOT DIGITS, R3	2095
						53	CE	000C2	MNEGL	R3, @CEXP	
			64	BE		30	3B	000C6	SKPC	#48, A_LENGTH, @A_ADDR	2104
						02	12	000CC	BNEQ	8\$	
						51	D4	000CE	CLRL	R1	
			1C	AE		51	DO	000D0	MOVL	R1, TEMP	2105
						50	D4	000D4	CLRL	R0	
						51	D5	000D6	TSTL	R1	
						02	12	000D8	BNEQ	9\$	
						50	D6	000DA	INCL	R0	
			24	AE		50	DO	000DC	MOVL	R0, STATUS	2106
						0F	12	000E0	BNEQ	10\$	2109
			50	AE	64	51	C3	000E2	SUBL3	R1, A_ADDR, R0	
					6A	50	A0	000E7	ADDW2	R0, A_LENGTH	
					64	51	DO	000EB	MOVL	R1, A_ADDR	2110
						03	11	000EF	BRB	11\$	2106

5C	BE	62	AE	24	BC	D4	000F1	10\$:	CLRL	@CSIGN	2113
					30	3B	000F4	11\$:	SKPC	#48, B_LENGTH, @B_ADDR	2115
					02	12	000FA		BNEQ	12\$	
		1C	AE		51	D4	000FC		CLRL	R1	
			52	1C	51	D0	000FE	12\$:	MOVL	R1, TEMP	
					AE	D0	00102		MOVL	TEMP, R2	2116
					50	D4	00106		CLRL	R0	
					52	D5	00108		TSTL	R2	
					02	12	0010A		BNEQ	13\$	
		24	AE		50	D6	0010C		INCL	R0	
			01	24	50	D0	0010E	13\$:	MOVL	R0, STATUS	
					AE	D1	00112		CMPL	STATUS, #1	2117
					0D	12	00116		BNEQ	14\$	
					8F	DD	00118		PUSHL	#STR\$ DIVBY ZER	2119
	00000000G		00		01	FB	0011E		CALLS	#1, LIB\$STOP	
50		5C	AE		52	C3	00125	14\$:	SUBL3	R2, B_ADDR, R0	2120
		62	AE		50	A0	0012A		ADDW2	R0, B_LENGTH	
		5C	AE		52	D0	0012E		MOVL	R2, B_ADDR	2121
			50	6A	AE	3C	00132		MOVZWL	A_LENGTH, R0	2126
			50	08	BC	C0	00136		ADDL2	@AEXP, R0	
			51	62	AE	3C	0013A		MOVZWL	B_LENGTH, R1	
			51	14	BC	C0	0013E		ADDL2	@BEXP, R1	
			50		51	C2	00142		SUBL2	R1, R0	
			50		53	C0	00145		ADDL2	R3, R0	2127
38	AE		50	20	BC	C1	00148		ADDL3	@RND TRUNC, R0, Q_LENGTH	
		08	AE	5A	AE	3C	0014E		MOVZWL	C_LENGTH, 8(SP)	2136
				38	AE	D5	00153		TSTL	Q_LENGTH	2129
					29	18	00156		BGEQ	16\$	
				18	AE	D4	00158		CLRL	LEADING ZEROS	2135
			50	08	AE	D0	0015B		MOVL	8(SP), R0	2136
					03	12	0015F		BNEQ	15\$	
			50		01	D0	00161		MOVL	#1, R0	
		10	AE		50	D0	00164	15\$:	MOVL	R0, BYTES_VM	
				54	AE	9F	00168		PUSHAB	START_BUF	2137
				14	AE	9F	0016B		PUSHAB	BYTES_VM	
	00000000G		00		02	FB	0016E		CALLS	#2, LIB\$GET_VM	
		24	AE		50	D0	00175		MOVL	R0, STATUS	
		2C	AE	14	AE	9E	00179		MOVAB	STORAGE, QSTRBUF	2138
			50		026F	31	0017E		BRW	29\$	2129
			50	62	AE	3C	00181	16\$:	MOVZWL	B_LENGTH, R0	2146
		50	AE	38	BE40	9E	00185		MOVAB	@B_LENGTH[R0], A_LEN	
			50		0E	C1	0018B		ADDL3	#14, A_LEN, R0	2151
4C	50		50		0F	C7	00190		DIVL3	#15, R0, A_CHUNKS	
	AE		50	62	AE	3C	00195		MOVZWL	B_LENGTH, R0	2152
			50		0E	C0	00199		ADDL2	#14, R0	
44	AE		50		0F	C7	0019C		DIVL3	#15, R0, B_CHUNKS	
	50	38	AE		1D	C1	001A1		ADDL3	#29, Q_LENGTH, R0	2153
34	AE		50		0F	C7	001A6		DIVL3	#15, R0, Q_CHUNKS	
	58	44	AE	34	AE	C1	001AB		ADDL3	Q_CHUNKS, B_CHUNKS, R8	2157
			50	4C	AE	D0	001B1		MOVL	A_CHUNKS, R0	
			58		50	D1	001B5		CMPL	R0, R8	
					03	1E	001B8		BGEQU	17\$	
			50		58	D0	001BA		MOVL	R8, R0	
		4C	AE		50	D0	001BD	17\$:	MOVL	R0, A_CHUNKS	
			51	44	AE	D0	001C1		MOVL	B_CHUNKS, R1	2163
			50	4C	BE41	3E	001C5		MOVAB	@A_CHUNKS[R1], R0	
10	AE		50		03	78	001CA		ASHL	#3, R0, BYTES_VM	

57	10	AE	18	C0	001CF	ADDL2	#24, BYTES VM	2166
	34	AE	OF	C5	001D3	MULL3	#15, Q_CHUNKS, R7	
51	1C	AE	0F	A7	9E	MOVAB	15(R7), TEMP	2168
	1C	AE	08	C1	001DD	ADDL3	8(SP), TEMP, R1	
		50	10	AE	D0	MOVL	BYTES VM, R0	
		51		50	D1	CMPL	R0, RT	
				03	1E	BGEQU	18\$	
	10	50	51	D0	001EA	MOVL	R1, R0	
		AE	50	D0	001EC	MOVL	R0, BYTES VM	
			54	AE	9F	PUSHAB	START_BUF	2177
			14	AE	9F	PUSHAB	BYTES VM	
	00000000G	00	02	FB	001F9	CALLS	#2, LIB\$GET_VM	
	24	AE	50	D0	00200	MOVL	R0, STATUS	
	3C	AE	54	AE	D0	MOVL	START_BUF, DRBUF	2178
48	AE	30	3C	AE	D0	MOVL	DRBUF, QBUF	2179
		50	08	C1	0020E	ADDL3	#8, QBUF, ABUF	2180
		5B	48	AE	D0	MOVL	ABUF, R11	2181
		50	4C	AE	D0	MOVL	A_CHUNKS, R0	
	40	AE	08	AB40	7E	MOVAQ	8(R11)(R0), BBUF	
		50	44	AE	D0	MOVL	B_CHUNKS, R0	2182
	28	AE	40	BE40	7E	MOVAQ	8BBUF(R0), QBBUF	
2C	AE	54	10	AE	C1	ADDL3	BYTES VM, START_BUF, R0	2183
		50	57	C3	00232	SUBL3	R7, R0, QSTRBUF	
		59	08	AB	9E	MOVAB	8(R11), R9	2187
		58	4C	AE	D0	MOVL	A_CHUNKS, R8	
		57	6A	AE	3C	MOVZWL	A_LENGTH, R7	
		56	64	AE	D0	MOVL	A_ADDR, R6	
6B	F0AD	CF	00000000G	00	16	JSB	LIB\$SCVT_STR_PACK_R9	
		59	40	AE	D0	MOVP	#15, ZERO, (R11)	2188
		58	44	AE	D0	MOVL	BBUF, R9	2189
		57	62	AE	3C	MOVL	B_CHUNKS, R8	
		56	5C	AE	D0	MOVZWL	B_LENGTH, R7	
			00000000G	00	16	MOVL	B_ADDR, R6	
		01	44	AE	D1	JSB	LIB\$SCVT_STR_PACK_R9	
			20	AE	13	CMPL	B_CHUNKS, #1	2195
		57	3C	AE	D4	BEQL	20\$	
		56	40	AE	D0	CLRL	FLAG	2198
			00000000G	00	16	MOVL	DRBUF, R7	2199
	24	AE	50	D0	0027A	MOVL	BBUF, R6	
		01	24	AE	D1	JSB	LIB\$CALC_D_R7	
		5A	08	AB	9E	MOVL	R0, STATUS	
50	4C	AE	36	13	00288	CMPL	STATUS, #1	2200
		59	60	9E	0028A	BEQL	19\$	
		57	08	AB	9E	MOVAB	8(R11), R10	2204
		58	4C	AE	D0	ADDL3	#1, A_CHUNKS, R0	
		56	3C	AE	D0	MOVAB	(R0), R9	
			00000000G	00	16	MOVL	8(R11), R7	2203
		5A	40	AE	D0	MOVL	A_CHUNKS, R8	
		59	44	AE	D0	MOVL	DRBUF, R6	
		57	40	AE	7D	JSB	LIB\$MUL_PACK_R10	
		56	3C	AE	D0	MOVL	BBUF, R10	2205
			00000000G	00	16	MOVL	B_CHUNKS, R9	
				0C	11	MOVQ	BBUF, R7	
6B	F03A	CF	OF	34	002C0	MOVL	DRBUF, R6	
						JSB	LIB\$MUL_PACK_R10	2200
						BRB	21\$	2209
						MOVP	#15, ZERO, (R11)	

04	AE	20	AE	04	11	002C6	BRB	21\$	2195
		34	AE	01	D0	002C8	MOVL	#1, FLAG	2212
		04	AE	03	78	002CC	ASHL	#3, Q_CHUNKS, 4(SP)	2216
	5A	28	AE	08	C2	002D2	SUBL2	#8, 4TSP)	
OC	AE	44	AE	08	C1	002D6	ADDL3	#8, QBBUF, R10	2229
			AE	01	C1	002DB	ADDL3	#1, B_CHUNKS, 12(SP)	
			6E	08	CE	002E1	MNEGL	#8, J	
				0081	31	002E4	BRW	24\$	
		50		AE	D0	002E7	MOVL	QBUF, R0	2221
59		50	30	6E	C1	002EB	ADDL3	J, R0, R9	
57		5B		6E	C1	002EF	ADDL3	J, R11, R7	
		58	20	AE	D0	002F3	MOVL	FLAG, R8	
		56	40	AE	D0	002F7	MOVL	BBUF, R6	
			00000000G	00	16	002FB	JSB	LIB\$\$CALC Q_R9	
		24	AE	50	D0	00301	MOVL	R0, STATUS	
			01	AE	D1	00305	CMPL	STATUS, #1	2222
				0D	13	00309	BEQL	23\$	
			00000000G	8F	DD	0030B	PUSHL	#LIB\$ INVARG	2224
				01	FB	00311	CALLS	#1, LIB\$STOP	
		56	50	AE	D0	00318	MOVL	QBUF, R0	2228
			50	6E	C1	0031C	ADDL3	J, R0, R6	
			59	AE	D0	00320	MOVL	12(SP), R9	
			57	AE	7D	00324	MOVQ	BBUF, R7	
				00	16	00328	JSB	LIB\$\$MUL PACK_R10	
57		5B	00000000G	6E	C1	0032E	ADDL3	J, R11, R7	2230
		58	28	AE	D0	00332	MOVL	QBBUF, R8	
		56	44	AE	D0	00336	MOVL	B_CHUNKS, R6	
			00000000G	00	16	0033A	JSB	LIB\$\$SUB PACK_R8	
		24	AE	50	D0	00340	MOVL	R0, STATUS	
			01	AE	D1	00344	CMPL	STATUS, #1	2234
				1E	12	00348	BNEQ	24\$	
			50	AE	D0	0034A	MOVL	QBUF, R0	2236
59		50	30	6E	C1	0034E	ADDL3	J, R0, R9	
50		6E		08	C1	00352	ADDL3	#8, J, R0	
57		5B		50	C1	00356	ADDL3	R0, R11, R7	
		58	40	AE	D0	0035A	MOVL	BBUF, R8	
		56	44	AE	D0	0035E	MOVL	B_CHUNKS, R6	
			00000000G	00	16	00362	JSB	LIB\$\$ADJUST Q_R9	
FF78	6E	08	04	AE	F1	00368	ACBL	4(SP), #8, J, -22\$	2216
		01	20	BC	D1	0036F	CMPL	@RND_TRUNC, #1	2241
				2C	12	00373	BNEQ	25\$	
57	34	AE		0F	C5	00375	MULL3	#15, Q_CHUNKS, R7	2244
		57	38	AE	C2	0037A	SUBL2	Q_LENGTH, R7	
	1C	AE	F1	A7	9E	0037E	MOVAB	-T5(R7), TEMP	
			38	AE	D7	00383	DECL	Q_LENGTH	2245
50	34	AE		03	78	00386	ASHL	#3, Q_CHUNKS, R0	2246
		50	30	AE	C0	0038B	ADDL2	QBUF, R0	
	3C	AE		70	7E	0038F	MOVAB	-(R0), DRBUF	
		57	1C	AE	D0	00393	MOVL	TEMP, R7	2247
		56	3C	AE	D0	00397	MOVL	DRBUF, R6	
			00000000G	00	16	0039B	JSB	LIB\$\$ROUND R7	
EF57	CF	30	BE	0F	35	003A1	CMPP3	#15, @QBUF, ZERO	2256
				54	DC	003A8	MOVPSL	R4	
54				02	EF	003AA	EXTZV	#2, #2, R4, R4	
	24	AE	01	54	C3	003AF	SUBL3	R4, #1, STATUS	
				06	12	003B4	BNEQ	26\$	2257
		18	AE	0F	D0	003B6	MOVL	#15, LEADING_ZEROS	2259

	EF44	CF	30	BE	26	11	003BA	26\$:	BRB	28\$		
					0F	35	003BC		CMPP3	#15, @QBUF, TEN	2262	
54		54		02	54	DC	003C3		MOVPSL	R4		
	24	AE		01	02	EF	003C5		EXTZY	#2, #2, R4, R4		
					54	C3	003CA		SUBL3	R4, #1, STATUS		
					09	18	003CF		BGEQ	27\$	2263	
			38		AE	D6	003D1		INCL	Q_LENGTH	2266	
			18	AE	0E	D0	003D4		MOVL	#14, LEADING_ZEROS	2267	
					08	11	003D8		BRB	28\$	2263	
			38	AE	02	C0	003DA	27\$:	ADDL2	#2, Q_LENGTH	2271	
			18	AE	0D	D0	003DE		MOVL	#13, LEADING_ZEROS	2272	
				58	AE	D0	003E2	28\$:	MOVL	QSTRBUF, R8	2279	
				56	AE	7D	003E6		MOVQ	QBUF, R6		
					00	16	003EA		JSB	LIB\$CVT_PACK_STR_R8		
	50		2C	AC	03	C1	003F0	29\$:	ADDL3	#3, CDIGITS, R0	2286	
				02	60	91	003F5		CMPB	(R0), #2		
					36	13	003F8		BEQL	30\$		
	50		2C	AC	03	C1	003FA		ADDL3	#3, CDIGITS, R0	2287	
				0B	60	91	003FF		CMPB	(R0), #11		
					2C	13	00402		BEQL	30\$		
			38	AE	AE	D1	00404		CMPL	8(SP), Q_LENGTH	2288	
					25	15	00409		BLEQ	30\$		
		50		08	AE	C3	0040B		SUBL3	Q_LENGTH, 8(SP), R0	2291	
1C		AE		50	18	AE	C3	00411	SUBL3	LEADING_ZEROS, R0, TEMP		
					38	AE	D0	00417	MOVL	8(SP), Q_LENGTH	2292	
					2C	AE	C2	0041C	SUBL2	TEMP, QSTRBUF	2293	
					1C	AE	D5	00421	TSTL	TEMP	2294	
					0F	19	00424		BLSS	31\$		
1C	AE		30	6E	00	2C	00426		MOVC5	#0, (SP), #48, TEMP, @QSTRBUF	2296	
					2C	BE	0042C					
					05	11	0042E		BRB	31\$	2286	
			2C	AE	18	AE	C0	00430	30\$:	ADDL2	LEADING_ZEROS, QSTRBUF	2299
					38	AE	D5	00435	31\$:	TSTL	Q_LENGTH	2312
					13	14	00438		BGTR	32\$		
					2C	AC	DD	0043A	PUSHL	CDIGITS	2314	
			10	AE	30	D0	0043D		MOVL	#48, 16(SP)		
					10	AE	9F	00441	PUSHAB	16(SP)		
			10	AE	01	D0	00444		MOVL	#1, 16(SP)		
					10	AE	9F	00448	PUSHAB	16(SP)		
					09	11	0044B		BRB	33\$		
					2C	AC	DD	0044D	32\$:	PUSHL	CDIGITS	2316
					30	AE	DD	00450	PUSHL	QSTRBUF		
					40	AE	9F	00453	PUSHAB	Q_LENGTH		
		00000000G	00		03	FB	00456	33\$:	CALLS	#3, LIB\$COPY_R_DX		
		24	AE		50	D0	0045D		MOVL	R0, STATUS		
					54	AE	9F	00461	PUSHAB	START_BUF	2317	
					14	AE	9F	00464	PUSHAB	BYTES_VM		
		00000000G	00		02	FB	00467		CALLS	#2, LIB\$FREE_VM		
		24	AE		50	D0	0046E		MOVL	R0, STATUS		
					04	00472			RET		2318	

; Routine Size: 1139 bytes, Routine Base: _STR\$CODE + 0D01

: 2239 2319 1 ROUTINE CHK_STR_TYPE (SRC_BUF, SRC_LEN, DST_DESC): NOVALUE =
: 2240 2320 1
: 2241 2321 1

```
2242 2322 1 1 ++
2243 2323 1 1
2244 2324 1 1 FUNCTIONAL DESCRIPTION:
2245 2325 1 1
2246 2326 1 1 This routine checks the destination string type and copies
2247 2327 1 1 the appropriate number of digits from the source buffer
2248 2328 1 1 into it as dictated by the rules of the destination string.
2249 2329 1 1
2250 2330 1 1 CALLING SEQUENCE:
2251 2331 1 1
2252 2332 1 1 CHK_STR_TYPE (src_buf.rnu.r,src_len.rl.r,dst_desc.wnu.dx)
2253 2333 1 1
2254 2334 1 1 FORMAL PARAMETERS:
2255 2335 1 1 SRC_BUF.rnu.r Addr of the source buffer which contains
2256 2336 1 1 the digits of the result.
2257 2337 1 1 SRC_LEN.rl.r Length of the source string
2258 2338 1 1 DST_DESC.wnu.dx Addr of the destination string (where to store
2259 2339 1 1 the resultant string).
2260 2340 1 1
2261 2341 1 1 IMPLICIT INPUTS:
2262 2342 1 1
2263 2343 1 1 -NONE
2264 2344 1 1
2265 2345 1 1 IMPLICIT OUTPUTS:
2266 2346 1 1
2267 2347 1 1 DST_DESC Store the resultant string in DST_DESC
2268 2348 1 1
2269 2349 1 1 ROUTINE VALUE:
2270 2350 1 1
2271 2351 1 1 -NONE
2272 2352 1 1
2273 2353 1 1 COMPLETION CODES:
2274 2354 1 1
2275 2355 1 1 -NONE
2276 2356 1 1
2277 2357 1 1 MACROS:
2278 2358 1 1
2279 2359 1 1 -NONE
2280 2360 1 1
2281 2361 1 1 SIDE EFFECTS:
2282 2362 1 1
2283 2363 1 1 -NONE
2284 2364 1 1 -
```



```
2286 2365 2 BEGIN
2287 2366
2288 2367 MAP
2289 2368 DST_DESC:REF BLOCK [8,BYTE];
2290 2369
2291 2370 LOCAL
2292 2371 TMP_BUF:REF VECTOR[65535,BYTE],
2293 2372 RESULT_DIGITS,
2294 2373 RLEN;
2295 2374
2296 2375 TMP_BUF = .SRC_BUF;
2297 2376 RESULT_DIGITS = ..SRC_LEN;
2298 2377
2299 2378 !+
2300 2379 Check the class of strings we are dealing with. For
2301 2380 dynamic and varying strings, return the calculated length;
2302 2381 for all other classes of strings, return the number of
2303 2382 characters specified in the destination descriptor.
2304 2383
2305 2384 IF (.DST_DESC[DSC$B_CLASS] EQL DSC$K_CLASS_D) OR
2306 2385 (.DST_DESC[DSC$B_CLASS] EQL DSC$K_CLASS_VS)
2307 2386 THEN
2308 2387 BEGIN
2309 2388 IF .DST_DESC[DSC$B_CLASS] EQL DSC$K_CLASS_VS ! Varying string only
2310 2389 THEN
2311 2390 BEGIN
2312 2391 RLEN = .DST_DESC[DSC$W_MAXSTLEN]; ! Fetch max string size
2313 2392 IF .RLEN LSS .RESULT_DIGITS ! If max < actual then
2314 2393 THEN !/return max # of chars
2315 2394 BEGIN
2316 2395 CHSMOVE (.RLEN,TMP_BUF,.DST_DESC[DSC$A_POINTER] + 2);
2317 2396 (.DST_DESC[DSC$A_POINTER])<0,16> = .RLEN;
2318 2397 END
2319 2398
2320 2399 (.DST_DESC[DSC$A_POINTER])<0,16> = .RLEN
2321 2400 ELSE
2322 2401 BEGIN ! Just retn # of
2323 2402 !/calculated characters
2324 2403 CHSMOVE (.RESULT_DIGITS,TMP_BUF,.DST_DESC[DSC$A_POINTER] + 2);
2325 2404 (.DST_DESC[DSC$A_POINTER])<0,16> = .RESULT_DIGITS;
2326 2405 END;
2327 2406
2328 2407 (.DST_DESC[DSC$A_POINTER])<0,16> = .RESULT_DIGITS;
2329 2408 STR$COPY_R (.DST_DESC,RLEN,.TMP_BUF);
2330 2409 END
2331 2410 ELSE
2332 2411
2333 2412 !+
2334 2413 Here we know the string is dynamic.
2335 2414 Return actual number of characters as calculated in algorithm.
2336 2415
2337 2416
2338 2417 STR$COPY_R (.DST_DESC,RESULT_DIGITS,.TMP_BUF);
2339 2418 END
2340 2419
2341 2420 !+
2342 2421 Here we know we are dealing with static strings.
```

```
2343  
2344  
2345  
2346  
2347  
2348  
2349  
2350  
2351  
2352  
2353  
2354  
2355  
2356  
2357  
2358  
2359  
2360  
2361  
2362  
2363  
2364  
2365  
2366  
2367  
2368  
2369  
2370  
2371  
2372  
2373  
2374  
2375  
2376  
2377
```

```
2422  
2423  
2424  
2425  
2426  
2427  
2428  
2429  
2430  
2431  
2432  
2433  
2434  
2435  
2436  
2437  
2438  
2439  
2440  
2441  
2442  
2443  
2444  
2445  
2446  
2447  
2448  
2449  
2450  
2451  
2452  
2453  
2454  
2455  
2456
```

```
3  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
4  
3  
4  
4  
4  
3  
2  
1
```

```
END;
```

```
!-  
ELSE  
  BEGIN  
    RLEN = .DST_DESC[DSC$W_LENGTH];  
    IF .RLEN GTR .RESULT_DIGITS  
    THEN  
      BEGIN  
        ! Fetch length passed  
        ! /in output descriptor  
        ! Given length>actual?  
        ! Yes.  
        Duplicate the zero character for the length  
        of the string. Then copy the calculated  
        numeric string into the appropriate offset into  
        the destination descriptor.  
        STR$DUPL CHAR (.DST_DESC,RLEN,%REF(%ASCII'0'));  
        CH$MOVE T.RESULT_DIGITS,.TMP_BUF,.DST_DESC[DSC$A_POINTER] +  
        .RLEN -.RESULT_DIGITS);  
      END  
      !+  
      ! Still dealing with static strings here.  
    ELSE  
      BEGIN  
        !+  
        ! For case where RLEN is less than or equal to the  
        ! actual length of the result, just copy RLEN digits  
        ! into the output descriptor.  
        STR$COPY_R (.DST_DESC,RLEN,.TMP_BUF);  
      END;  
    END;  
  END;
```

003C 00000 CHK_STR_TYPE:						
	5E		0C C2 00002	.WORD	Save R2,R3,R4,R5	2319
	53	04	AC D0 00005	SUBL2	#12, SP	
04	AE	08	BC D0 00009	MOVL	SRC BUF, TMP BUF	2375
	52	0C	AC D0 0000E	MOVL	@SRC LEN, RESULT_DIGITS	2376
	02	03	A2 91 00012	MOVL	DST_DESC, R2	2384
			06 13 00016	CMPB	3(R2), #2	
	0B	03	A2 91 00018	BEQ	1\$	
			26 12 0001C	CMPB	3(R2), #11	2385
	0B	03	A2 91 0001E	BNEQ	4\$	
			19 12 00022	CMPB	3(R2), #11	2388
08	AE		62 3C 00024	BNEQ	3\$	
04	AE	08	AE D1 00028	MOVZWL	(R2), RLEN	2391
			07 18 0002D	CMPL	RLEN, RESULT_DIGITS	2392
04	B2	08	AE B0 0002F	BGEQ	2\$	
			3A 11 00034	MOVW	RLEN, @4(R2)	2399
				BRB	5\$	

STR\$ARITH
1-019

B 16
16-Sep-1984 01:27:51 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:40:01 [LIBRTL.SRC]STRARITH.B32;1

Page 72
(16)

04	B2	04	AE	B0	00036	2\$:	MOVW	RESULT_DIGITS, @4(R2)	:	2407
			33	11	0003B		BRB	5\$:	2408
			53	DD	0003D	3\$:	PUSHL	TMP_BUF	:	2417
		08	AE	9F	0003F		PUSHAB	RESULT_DIGITS	:	
			31	11	00042		BRB	6\$:	
08	AE		62	3C	00044	4\$:	MOVZWL	(R2), RLEN	:	2426
04	AE	08	AE	D1	00048		CMPL	RLEN, RESULT_DIGITS	:	2428
			21	15	0004D		BLEQ	5\$:	
	6E		30	DD	0004F		MOVL	#48, (SP)	:	2437
			5E	DD	00052		PUSHL	SP	:	
		0C	AE	9F	00054		PUSHAB	RLEN	:	
			52	DD	00057		PUSHL	R2	:	
00000000G	00		03	FB	00059		CALLS	#3, STR\$DUPL_CHAR	:	
50	04	08	AE	C1	00060		ADDL3	RLEN, 4(R2), R0	:	2439
		04	AE	C2	00066		SUBL2	RESULT_DIGITS, R0	:	
60		04	AE	28	0006A		MOVCL	RESULT_DIGITS, (TMP_BUF), (R0)	:	
				04	0006F		RET		:	2428
			53	DD	00070	5\$:	PUSHL	TMP_BUF	:	2453
		0C	AE	9F	00072		PUSHAB	RLEN	:	
			52	DD	00075	6\$:	PUSHL	R2	:	
00000000G	00		03	FB	00077		CALLS	#3, STR\$COPY_R	:	
			04	0007E			RET		:	2456

; Routine Size: 127 bytes, Routine Base: _STR\$CODE + 1174

```
2379 2457 1 ROUTINE FREE_STRINGS (      | Free local strings
2380 2458 1     SIG,                      | Signal vector
2381 2459 1     MECH,                    | Mechanism vector
2382 2460 1     ENBL                  | Enable vector
2383 2461 1     ) =
2384 2462 1
2385 2463 1 ++
2386 2464 1 FUNCTIONAL DESCRIPTION:
2387 2465 1
2388 2466 1     If we are unwinding, free the local strings. They are passed
2389 2467 1     in the enable vector.
2390 2468 1
2391 2469 1 FORMAL PARAMETERS:
2392 2470 1
2393 2471 1     SIG.rl.a      A counted vector of parameters to LIB$SIGNAL/STOP
2394 2472 1     MECH.rl.a    A counted vector of info from CHF
2395 2473 1     ENBL.ra.a   A counted vector of ENABLE argument addresses.
2396 2474 1
2397 2475 1 IMPLICIT INPUTS:
2398 2476 1
2399 2477 1     NONE
2400 2478 1
2401 2479 1 IMPLICIT OUTPUTS:
2402 2480 1
2403 2481 1     NONE
2404 2482 1
2405 2483 1 ROUTINE VALUE:
2406 2484 1 COMPLETION CODES:
2407 2485 1
2408 2486 1     Always SS$_RESIGNAL, which is ignored when unwinding.
2409 2487 1
2410 2488 1 SIDE EFFECTS:
2411 2489 1
2412 2490 1     Frees all of the strings passed as enable arguments.
2413 2491 1
2414 2492 1 --
2415 2493 1
2416 2494 2 BEGIN
2417 2495 2
2418 2496 2 MAP
2419 2497 2     SIG : REF VECTOR,
2420 2498 2     MECH : REF VECTOR,
2421 2499 2     ENBL : REF VECTOR;
2422 2500 2
2423 2501 2 +
2424 2502 2 | Only free strings if this is the UNWIND condition.
2425 2503 2 -
2426 2504 2
2427 2505 2     IF ( NOT (LIB$MATCH_COND (SIG [1], %REF (SS$_UNWIND)))) THEN RETURN (SS$_RESIGNAL);
2428 2506 2
2429 2507 2 +
2430 2508 2 | Go through the enable arguments, freeing them.
2431 2509 2 -
2432 2510 2
2433 2511 2     INCR ARG_NO FROM 1 TO .ENBL [0] DO
2434 2512 2         IF (..ENBL [.ARG_NO] NEQ 0) THEN STR$FREE1_DX (.ENBL [.ARG_NO]);
2435 2513 2
```


STRARITH
1-019

E 16
16-Sep-1984 01:27:51
14-Sep-1984 12:40:01

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]STRARITH.B32:1

Page 75
(17)

COMMAND QUALIFIERS

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LISS:STRARITH/OBJ=OBJ\$:STRARITH MSRC\$:STRARITH/UPDATE=(ENH\$:STRARITH)

; Size: 4324 code + 328 data bytes
; Run Time: 00:48.5
; Elapsed Time: 03:07.9
; Lines/CPU Min: 3113
; Lexemes/CPU-Min: 19754
; Memory Used: 373 pages
; Compilation Complete

0213 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY